Lecture Notes in
Computer Science 960

Daniel Leivant (Ed.)

Logic and Computational Complexity

International Workshop LCC '94
Indianapolis, IN, USA, October 1994
Selected Papers

Springer

Lecture Notes in Computer Science 960

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer

Berlin
Heidelberg
New York
Barcelona
Budapest
Hong Kong
London
Milan
Paris
Tokyo

Daniel Leivant (Ed.)

Logic and Computational Complexity

International Workshop LCC '94
Indianapolis, IN, USA, October 13-16, 1994
Selected Papers

Springer

Series Editors

Gerhard Goos
Universität Karlsruhe
Vincenz-Priessnitz-Straße 3, D-76128 Karlsruhe, Germany

Juris Hartmanis
Department of Computer Science, Cornell University
4130 Upson Hall, Ithaca, NY 14853, USA

Jan van Leeuwen
Department of Computer Science, Utrecht University
Padualaan 14, 3584 CH Utrecht, The Netherlands

Volume Editor

Daniel Leivant
Department of Computer Science, Indiana Univesity
Lindley Hall 215, Bloomington, IN 47405-4101, USA

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Logic and computational complexity : international workshop ; selected papers / LCC '94, Indianapolis, IN, USA, October 13 - 16, 1994. Daniel Leivant (ed.). - Berlin ; Heidelberg ; New York : Springer, 1995
(Lecture notes in computer science ; 960)
ISBN 3-540-60178-3
NE: Leivant, Daniel [Hrsg.]; LCC <1994, Indianapolis, Ind.>; GT

CR Subject Classification (1991): F.1.3, F.2, F.4.1

1991 Mathematics Subject Classification: 03Cxx, 03Fxx

ISBN 3-540-60178-3 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10486541 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

The synergy between logic and computational complexity has gained importance and vigor in recent years, cutting across areas such as proof theory, finite model theory, computation theory, applicative programming, database theory, and philosophical logic. This volume is the outcome of a Workshop on Logic and Computational Complexity (LCC), organized to bring together researchers in this growing interdisciplinary field, so as to foster and enhance collaborations and to facilitate the discovery of conceptual bridges and unifying principles.

The workshop was held at the Indiana University Conference Center in Indianapolis, from the 13th to the 16th of October, 1994. Forty-one talks were presented at the meeting. The papers in this volume are revised versions of the workshop papers, or closely related to them. Jean-Yves Girard, Stephen Lindell and Anil Seth were unable to attend, but included their intended contribution in this volume. On the other hand, a number of speakers could not contribute to this volume, for various reasons: Maria Bonet, Philip Bradford, Kevin Compton, Stephen Cook, Alexandar Ignjatovic, Neil Immerman, Phokion Kolaitis, Harry Mairson, Yiannis Moschovakis, Jim Otto, Toni Pitassi, Andre Scedrov, Wilfried Sieg, Alexei Stolboushkin, Alasdair Urquhart, Dirk Van Gucht, Moshe Vardi, and Victor Vianu. Other participants who graciously agreed to chair sessions were Serge Abiteboul, Paris Kanellakis, Julia Knight, Larry Moss, Jim Royer, and Jouko Vaananen.

The papers in this volume are grouped by themes, but the frequent difficulty of classifying papers demonstrates in itself the close relations between various approaches. While contributions were not refereed, an editorial effort has been made to provide authors with informal feedback from readers, which in many cases proved indeed to be very constructive. With apologies to the many whose help has not been properly recorded, I would like to thank for their feedback Stephen Bellantoni, Stephen Bloch, Sam Buss, Peter Clote, Anuj Dawar, Georg Gottlob, Hagen Huwig, Paris Kanellakis, Max Kelly, Phokion Kolaitis, Julia Knight, Janos Makowsky, Anil Nerode, Helmut Schwichtenberg, Philip Scott, Robert Seely, Alasdair Urquhart, Moshe Vardi, and Victor Vianu.

LCC was organized with help from my Indiana University colleague and friend Larry Moss, to whom I am most grateful. The meeting could not have taken place without generous grants from the Office of Naval Research, the National Science Foundation, the Indiana University Interdisciplinary Venture Fund, and the Dean of the Indiana University School of Arts and Sciences. Speaking for all participants, I am sincerely obliged to these supporters, not only for their generosity, but also for their show of confidence in the significance of this growing research community.

Daniel Leivant
Indiana University
May 1995

Table of Contents

Foundational Issues

Applicative and Proof Theoretic Complexity

Complexity of Proofs

Computational Complexity of Functionals

Complexity and Model Theory

Finite Model Theory

Strict Finitism and Feasibility

Felice Cardone*

Università di Milano

Abstract. We examine the relation between predicative recurrence and strictly finitistic tenets in the philosophy of mathematics, primarily by focusing on the rôle of numerical notations in computing. After an overview of Wittgenstein's ideas on the "surveyability" of notations, we analyze a subtle form of circularity in the usual justification of the primitive recursive definition of exponentiation (Isles 1992), and suggest connections with recent works on predicative recurrence (Leivant 1993b, Bellantoni & Cook 1993).

A long-standing thread in the foundations of mathematics questions the ontology of numeric terms and the absoluteness of the informal notion of finiteness. This paper is a preliminary exploration of some of the ideas (and puzzles) underlying these "strictly finitistic" approaches, especially in the light of recent works that relate predicative formalisms to computational complexity (Nelson 1986, Buss 1986, Leivant 1993a). One would expect investigations in these areas to lead to a unified perspective, thereby contributing to an assessment of the Karp-Cook Thesis on identifying feasible computability with poly-time (see (Davis 1982); this identification goes back to Edmonds 1965).

Large finite collections, and the natural numbers that count them, are the main source of perplexity for strict finitists: their acceptance makes the ontological status of constructions requiring an arbitrarily large but finite number of steps indistinguishable from that of the objects of platonistic mathematics. We shall

[*] Address: Dipartimento di Scienze dell'Informazione, via Comelico 39/41, I-20135 Milano, Italy. E-mail: `felice@imiucca.csi.unimi.it`. This is an expanded version of my talk for the Logic and Computational Complexity Meeting, and is part of a larger project analyzing the confluence of mathematical results and philosophical arguments in the analysis of feasible computations that will be described in a future joint paper with Daniel Leivant, to whom I am greatly indebted for stimulating my interest in this field and for prompting me to write this preliminary account. It will be apparent from the text that his views, and also those of David Isles, have been quite influential on the present paper (of course they are not at all responsible for any of my mistakes). I am grateful also to Stephen Bellantoni, Paolo Boldi, Gabriele Lolli, Diego Marconi, Piergiorgio Odifreddi and Nicoletta Sabadini for criticisms and suggestions, and to Roberta Mari for playing both as Proponent and Opponent in the design of the games described in the last section. Finally, I would like to thank my friend, Miss Claudia Bonino, for betting long ago that this paper would eventually have been written. The research was supported by CNR Cooperation Project "Linguaggi Applicativi e Dimostrazioni Costruttive" and MURST 40%.

make no attempt here to work out a unified account of strict finitism, and limit ourselves to a brief overview of the web of authors who contributed to it in some way. Early advances were made by some of the French pre-intuitionists, especially Émile Borel, whose position is developed at some length in (Borel 1952), and independently by Gerritt Mannoury (1909, 1931). Their ideas were the starting point of the reflections of van Dantzig, who is cited by Esenin-Vol'pin (1961, 1970) together with Fréchet and Lusin.[2] Bernays (1935) was probably the first to realize that a notion of *practical* possibility, associated to the conception of mathematics as a human activity, might form the basis of one possible position in the foundations of mathematics. If this is taken as defining strict finitism (as (Wang 1958), (Kreisel 1958) and also (Engeler 1971), in his mathematical model of strict finitism, seem to suggest), then strict finitism is reduced to an extreme form of constructivism, which rules out constructions that cannot be performed *in practice* as well as mathematical statements whose proof depends on such constructions. Later on, the works of van Dantzig and Esenin-Vol'pin complemented this interpretation and explicitly put forward a position that has provided much of the basis for further developments in this area. These include the work of van Bendegem (1985); the formalization and analysis of parts of Esenin-Vol'pin's work carried out by Geiser (1974) and Isles (1981), and the papers by Isles (1992, 199?). A formal analysis of feasibility is described in (Parikh 1971), and by different means in (Simon 1977). Sazonov (1992) builds on Parikh's work, and also suggests an interesting connection with the Alternative Set Theory of Vopěnka (1979), which was originally intended also to contribute to the formalization of vagueness (Sochor 1984) and explicitly indicated by Vopěnka to be applicable in particular to the treatment of vague concepts employed by Esenin-Vol'pin (1961) in his construction of non-isomorphic natural number series. On the purely philosophical side, the vagueness of the concept of feasibility has originated a series of investigations that start from Dummett's analysis of paradoxes of the Sorites type (Dummett 1975), and culminates with (Wright 1982). Finally, in a very different philosophical framework, the emphasis on the limitations of mathematical activity interpreted as a human and (therefore) symbolic practice is central to Rotman's "corporeal semiotics" of mathematics (Rotman 1993).

A completely independent strictly finitistic picture of mathematical activity also emerges from the later philosophy of Wittgenstein. Kreisel, in his review of the *Remarks on the Foundations of Mathematics*, pointed out the similarity between some of Wittgenstein's ideas on proofs and calculations and a position which involves a distinction, ignored by Hilbert's finitism, between *constructions which consist of a finite number of steps and those which can actually be carried out, or between configurations which consist of a finite number of discrete parts and those which can actually be kept in mind (or surveyed)* (Kreisel 1958). It was later argued by Dummett (1959) that the emphasis on the practical notion of possibility that, according to Kreisel, is involved in the concept of surveyability, seems to be somewhat misplaced. What is really essential in Wittgenstein's arguments is instead the ramification of proof techniques that is necessary for

[2] Lusin's ideas have also been mentioned by Rashevskii (1973).

dealing with increasingly larger combinatorial configurations, numbers in particular, as opposed to configurations that can be easily taken in. This position necessarily gives a prominent place to the investigation of the rôle played by notations in computing, and will be taken in the present paper as one main consequence of the strict finitist attitude. After an overview of Wittgenstein's remarks on surveyability, stressing the cognitive aspects of notations, we shall turn in a later section to an orthogonal – though not unrelated – analysis of notations pertaining to their operational behavior and therefore more amenable to formal treatment by means of standard techniques, inspired by recent works of Isles (1992), Leivant (1993b) and Bellantoni & Cook (1993).

1 Surveyability

> If you have a proof-pattern that cannot be taken in,
> and by a change in notation you turn it into one that can,
> then you are producing a proof, where there was none before.
> (Wittgenstein 1956, II 2)

The distinction between surveyable and unsurveyable configurations is central in Wittgenstein's critique of Russell's purely logical reconstruction of arithmetic. The logicist project does not *reduce* arithmetic to logic because the reliability of a proof hinges on a criterion which is external to the calculus itself:

> *Imagine that you had written down a 'formula' a mile long, and you shewed by transformation that it was tautologous ('if* it *has not altered meanwhile', one would have to say). Now we* count *the terms in the brackets or we divide them up and make the expression into one that can be taken in, and it comes out that there are* 7566 *terms in the first pair of brackets,* 2434 *in the second,* 10000 *in the third. Now have I proved that* 2434 + 7566 = 10000*?—That depends—one might say— on whether you are certain that the counting has really yielded the number of terms which stood between the brackets in the course of the proof.* (Wittgenstein 1956, II 7)

It is essential for arithmetical calculations not only the result but also the technique employed for arriving at it, and by translating these into logical proofs this intensional feature of calculations is lost. In fact,

> *[...] could we find out the truth of the proposition* 7034174 + 6594321 = 13628495 *by means of a proof carried out in the [stroke] notation?—Is there such a proof of this proposition?—The answer is: no.* (ibidem, II 3)

This happens in part also because the notation has changed the statement into one that is not surveyable, and there is no certainty that, in performing the proof, the marks on the paper do not change without our noticing it:

> *Imagine someone giving us a sum to do in a stroke-notation, say |||||||||| + |||||||||||, and, while we are calculating, amusing himself by removing and adding strokes without our noticing. He would keep on saying: "but the sum isn't right", and we would keep going through it again, fooled every time.—Indeed, strictly speaking, we wouldn't have any concept of a criterion for the correctness of the calculation.*
> (Wittgenstein 1969, IV 18, page 330)

Wittgenstein's remarks on the unsurveyability of long numerical signs, their failure to behave as paradigms (Wittgenstein 1956, II 7) or their lack of a characteristic pattern (ibidem, II 11) do not entail that signs like |||||||||| can never be used as tools for counting. In stone-age arithmetic pebbles were used to count the animals in a herd and recheck the count later. Such a stroke-like notation is also effective in comparing the size of two herds or, if the above check is the only purpose of the counting, in adding two such counts, but not for many other numerical operations in real life. The point in this example is that what is being performed here is not a calculation but rather an experiment, that is, something that is outside arithmetic:

> *[...] "Proof must be surveyable": this aims at drawing our attention to the difference between the concepts of 'repeating a proof', and 'repeating an experiment'. To repeat a proof means, not to reproduce the conditions under which a particular result was once obtained, but to repeat every step* and the result. *And although this shews that proof is something that must be capable of being reproduced* in toto *automatically, still every such reproduction must contain the force of proof, which compels acceptance of the result.* (ibidem, II 47,55)

By translating stroke numerals into decimal notation (that is, as one would normally say, by *counting* them) we sometimes recognize miscalculations, but this makes the former technique to depend on another one, and the criteria for establishing the identity of a proof external to the calculus in which the proof is carried out. For this reason it is important to insist that

> *'A mathematical proof must be perspicuous.' Only a structure whose reproduction is an easy task is called a "proof". It must be possible to decide with certainty whether we really have the same proof twice over, or not. The proof must be a configuration whose exact reproduction can be certain.* (ibidem, II 1)

This emphasis on surveyability clearly rests to a great extent on an understanding of mathematics as a human activity performed through the manipulation of *physical* signs (say, e.g., as marks located in the physical space or embodied in a limited memory). It is in such a perspective that one must be prepared to accept to place on this activity the natural limitations arising from the psychology of human cognitive processes:

> *beyond a certain length we cannot distinguish [unary numerals] any further without counting the strokes, and so without translating the signs*

> *into different ones. "||||||||||" and "|||||||||||" cannot be distinguished in the same sense as* 10 *and* 11, *and so they aren't in the same sense distinct signs.* (Wittgenstein 1969, IV 18, page 330)

There is a striking similarity between these arguments and those that are put forward in Turing's analysis of mechanical computation. In particular, it is the boundedness of our visual field that prevents us from recognizing at one glance the symbols 157767733443477 and 157767733443477 as being the same (Turing 1936, page 251), and Turing's analysis takes into account this impossibility by requiring that the symbols used by his devices be written in limited portions of the tape (the squares) and be immediately recognizable (ibidem; see also Kleene 1952, §70): infinitely many symbols, each of them occupying at most only a fixed, finite amount of space, would be indistinguishable and their encoding as strings of simple symbols would exceed, for infinitely many of them, that amount.[3] It is remarkable, then, that according to Wittgenstein one can sometimes make surveyable a long symbolic configuration by complicating it. So, for example,

> *[...] it is essential to the calculus with* 1000000 *that this number must be capable of resolution into a sum* $1+1+1\ldots$, *and in order to be certain that we have the right number of units before us, we can number the units:*
>
> $$\begin{array}{cccccc} 1 & +\,1 & +\,1 & +\,1 & \ldots & +\,1 \\ 1 & 2 & 3 & 4 & & 1000000. \end{array}$$
>
> *This notation would be like:* '100,000.000,000' *which also makes the numeral surveyable.* (Wittgenstein 1956, II 16)

Hence what *is* surveyable is not the concrete, spatial configuration consisting, say, of marks on paper, but rather its pattern, whose abstract character is shown by its independence of space, a non-additive behavior which reveals its nature of a *Gestalt*:

> *How can I know that* |||||||||| *and* |||||||||| *are the same sign? After all it is not enough that they look* alike. *For having roughly the same gestalt can't be what is to constitute the identity of the signs, but just their being the same in number.*
>
> *(The problem of the distinction between* $1+1+1+1+1+1+1$ *and* $1+1+1+1+1+1+1+1$ *is much more fundamental than appears at first sight. It is a matter of the distinction between physical and visual number.)* (Wittgenstein 1969, IV 18, page 331)

Observe that it is unlikely that this interpretation of surveyability may be reformulated in terms of devices operating in a manner sufficiently similar to that of

[3] Especially interesting in the present perspective is the formulation of the principles of mechanisms carried out by Gandy in terms of an analogy with physics, and based on the existence of a *lower bound on the linear dimensions of every atomic part of the device* and of an *upper bound (the velocity of light) on the speed of propagation of changes* (Gandy 1980, page 126).

Turing machines. As observed by Gandy (1980), *Gestalten* have a global character that is irreducible to the iteration of simple local steps proper to such mechanisms. Wittgenstein's examples, instead, exploit the fact that this can happen in our use of symbolic configurations when performing calculations or proofs. This is not in conflict with the existence of limitations upon our ability to recognize patterns: it simply shifts the emphasis on these as the true objects of attention when carrying out the symbolic manipulations involved in our non formalized proofs. Also, it is in this interpretation that the geometric properties of proofs are of a special relevance:

> *[...] logical proof, e.g. of the Russellian kind, is cogent only so long as it also possesses geometrical cogency. And an abbreviation of such a logical proof may have this cogency and so be a proof, when the Russellian construction, completely carried out, is not.*
>
> *[...] The consideration of* long *unsurveyable logical proofs is only a means of shewing how this technique—which is based on the geometry of proving—may collapse, and new techniques become necessary.* (Wittgenstein 1956, II 43,45)

Surveyability is just the aspect through which the geometrical cogency of proofs becomes manifest; admittedly, this is a vague property of symbolic configurations, and a common objection to according any relevance to it is that *seeing whether [a proof of a Russellian proposition stating an addition like '$a + b = c$', consisting of a few thousand signs] is correct or not is a purely external difficulty, of no mathematical interest. ("One man takes in easily what someone else takes in with difficulty or not at all" etc. etc.)* but this objection implicitly presupposes that *the definitions serve merely to abbreviate the expression for the convenience of the calculator; whereas they are part of the calculation. By their aid expressions are produced which could not have been produced without it* (ibidem, II 2). It appears at this point that the stress on surveyability, rather than being a study of practical limitations, takes these for granted in order to develop a *theory of meaning* for mathematical statements close in spirit to the theory of meaning underlying intuitionistic mathematics, as advocated by Dummett (1977). This was pointed out by Dummett himself in relation to Wittgenstein's ideas: *what the word 'prime' means as applied to large numbers is shown by what we accept as the* criterion *for primality, what we take as the standard whereby to assess claims that a number is prime or composite* (Dummett 1959, page 181). Notations do have a rôle in this perspective because their being surveyable or not may necessitate a change in the techniques for manipulating them, and it is by means of techniques that we *use* signs, and give them their meaning. Strict finitism is not a restriction of the acceptable constructions to those that are feasible, in particular strictly finitistic arithmetic is not that of feasible numbers, but a controlled use of notations that takes into account the fact that it is not always possible to pass from one notation to another without changing the method of counting, and thereby the meaning of the statement in which the notation occurs. This means in particular that it is legitimate from the point of view of (this interpretation of) strict finitism to use, for example, exponential notations:

> *one does want to use numbers like* 6.0229×10^{26} *(Avogadro's number), even if we cannot count up to it. We can write it down, add* 10^{12} *to it and write that down – in other words we can manipulate it.*
> (Simon 1977, page 195)

2 Notations and numbers

> What are numbers? – What numerals signify; an investigation of what they signify is an investigation of the grammar of numerals.
> (Wittgenstein 1969, IV 18, page 321)

Making explicit when two notations (or two techniques) are equivalent is a major problem arising from Wittgenstein's remarks, as pointed out already by Kreisel (1958). An especially interesting instance of this problem appears in an intensional counterpart of the totality proofs of recursive functions, that in a strictly finitistic context arises already in the case of functions defined by primitive recursion, notably exponentiation (Esenin-Vol'pin 1970), (van Dantzig 1956). Sometimes, when introducing a new notation by means of such a definition, we are more likely to introduce another way of counting that must be kept distinct from the counting by units implicit in the conception of numerals as the expressions obtained from $\mathbf{0}$ by application of $\mathbf{s}(\cdot)$.

The problematic status of primitive recursive definitions can be seen already from the fact that the usual set-theoretical justifications of primitive recursion follow the same line of the original one of Dedekind (1901) in using impredicative comprehension principles (see also (Henkin 1960)), and there seems to be an irreducible amount of impredicativity in any attempt to avoid introducing natural number series as processes (see (Parsons 1992) for a discussion of this aspect of the induction principle). Natural means for analyzing the ideas behind a genetic view of numbers, still preserving to a limited extent the possibility of inductive reasoning, lead directly to a logical characterizations of the class of functions computable in polynomial time in (Leivant 1993a), and provide a link between the theory of feasible computations and the foundational issues that are in the background of most strict finitist positions. This analysis also yields, as an important result, a notion of *tiering* which allows the description of feasibly computable functions by means of equational systems in which tiers are used as sorts that prevent the formation of (primitive recursive) functions which violate predicatively justifiable constraints (Bellantoni 1992, Bellantoni & Cook 1993 and Leivant 1993b).

Independently, an argument aimed at showing the existence of a subtle form of circularity in the ordinary proofs of totality of exponentiation has been recently developed by Isles (1992, 199?) in the framework of first-order theories, in particular Peano arithmetic (see also (Esenin-Vol'pin 1970, page 33) for a related claim). By dropping the assumption that variables range over the whole model of the theory, it becomes possible to set up a system for annotating the inference within a proof by means of "arrows" which describe the referential assumptions on the variables implicit in the justification of the correctness of each application of the rule. This analysis is intensional, in that different proofs of the same formula may lead to different assumptions on the range of the same variables ("reference grammars" in the terminology of (Isles 199?)). It turns out that the usual inductive proofs of totality of exponentiation require for their correctness that the underlying model be already closed under the exponentiation function. Both of these approaches emphasize, though in different forms, two symbolic levels involved in definitions of functions: on one hand we deal with basic expressions, whose numerical meaning is unproblematic; on the other hand, we introduce new notations whose reducibility to those of the first level has to be proved in some way, under specific assumptions.

Beside these, we propose below a natural operational analysis of primitive recursion that contributes to put in a formal setting at least some of the intensional aspects involved in transforming one notation into another, and also allows to single out some necessary conditions for the possibility of such transformations that we shall outline in some cases related to the usual primitive recursive algorithm for exponentiation, corresponding to those discussed in (Isles 1992). At the intensional level the totality of a function corresponds to the possibility of eliminating the defined symbol denoting it. Any expression in which a defined function symbol occurs gives rise to a reduction process directed to the elimination of it and of any other defined symbol contributing to its definition, hereditarily. Observing that primitive recursive definitions are instances of the orthogonal (= left-linear and without critical pairs) term rewriting systems (Klop 1990), we apply to our examples some techniques introduced in (Huet & Levy 1979) for analyzing computations in this class of systems. The main goal of our proposal, not achieved at present, will be to develop *semantical* counterparts of tiering that may suggest an alternative explanation of the circularities brought to light by Isles' analysis.

2.1 Redexes and derivations

We recall the basic notions needed for our analysis: most of them are standard concepts from the theory of term rewriting systems, and we refer to (Huet & Levy 1979) and (Klop 1990) for a thorough development. We shall consider **terms** built over **constructors** $\mathbf{0}$ and $\mathrm{s}(\cdot)$, using variables from a set $\mathcal{V}$ and symbols for defined n-ary functions ($n \geq 0$). Primitive recursion takes then the form:

$$\begin{aligned} F(\vec{x}, \mathbf{0}) &= M[\vec{x}] \\ F(\vec{x}, \mathrm{s}(y)) &= N[\vec{x}, y, F(\vec{x}, y)] \end{aligned}$$

where M, N are terms possibly containing occurrences of the subterms listed within square brackets. Primitive recursive definitions can thus be seen as examples of term rewriting system with constructors, in which there is a partition of function symbols into defined symbols $\mathcal{D}$ and constructors $\mathcal{C}$ and each rewriting rule has the form $F(M_1, \ldots, M_n) = N$, where $F \in \mathcal{D}$ and the only function symbols occurring in each M_i, for $i = 1, \ldots, n$, are in $\mathcal{C}$.

For many purposes it will be necessary to keep track of the position of a symbol, and this is done by means of "Dewey's decimal notation" (see Knuth 1968, pages 310 ff.): the **position** (sometimes also called **occurrence**) of a symbol in a term M is given by the sequence of (positive integers assigned from left to right to the) nodes along the unique path from the root to that symbol in the construction tree of the term, the root position being the empty sequence ϵ. These form the set $\mathcal{O}(M)$, which can be ordered by the relation $u \prec v$ which holds only if $u, v \in \mathcal{O}(M)$ and u is a prefix of v. If two positions u, v are incomparable in this ordering they are said to be disjoint, written $u \mid v$; if $u \prec v$ then their difference is $v - u$, the position which consists of the symbols of v except for the initial segment u. The subterm of M whose principal symbol is at position u is denoted by M/u: its positions are all the differences $v - u$ for $v \in \mathcal{O}(M)$ such that $v \succ u$. The root symbol of M/u is then $M(u)$, looking at a term as a (partial) function from positions to symbols. If a term M contains a subterm P at some unspecified position u, we can consider a new term $M'[\ ]$ which is like M except for having a special symbol $[\ ]$, a **hole**, at position u. $M'[\ ]$ is called a **context**, and M can be regarded as the term $M'[P]$, in which the hole has been filled with P.

A **redex** is any term R for which there exists a substitution σ of terms for variables such that $R = \sigma(\lambda)$, where $\lambda = \rho$ is one of the rules of the system; λ is said in this case to be the **redex scheme**. Given a term $M = N[R]$ containing such a redex as a subterm, an **elementary derivation**, written $M \rightarrow M'$, replaces $R = \sigma(\lambda)$ with its **contractum** $R' = \sigma(\rho)$, yielding $M' = N[R']$. If M/u is the redex R, then the elementary derivation contracting u is written $M \rightarrow_u M'$. By reflexive transitive closure we obtain a reduction relation $P \rightarrow^* Q$ which holds whenever there is a **derivation** starting at P and ending at Q, i.e. a sequence of terms $M_0, \ldots, M_n$ such that

$$P = M_0 \rightarrow_{u_0} M_1 \rightarrow_{u_1} \cdots \rightarrow_{u_{n-2}} M_{n-1} \rightarrow_{u_{n-1}} M_n = Q.$$

with each u_i belonging to the set $\mathcal{R}(M_i)$ of redex occurrences in the term M_i. A term containing no redex as a subterm is said to be in **normal form**. The preceding definitions can be generalized to **multiderivations** that at each step contract finitely many disjoint redex occurrences. The notation $M \Rightarrow^* N$ indicates the existence of a multiderivation with initial term M and final term N, composed of elementary multiderivations

$$M = M_0 \Rightarrow_{U_0} M_1 \Rightarrow_{U_1} M_2 \Rightarrow_{U_2} \cdots \Rightarrow_{U_{n-2}} M_{n-1} \Rightarrow_{U_n} M_n = N,$$

where $U_i \subseteq \mathcal{R}(M_i)$ for $i < n$. Clearly, derivations can be considered as multiderivation which at each step contract a singleton.

2.2 Residuals

It is important to have some way to keep track of the amount of work performed by a multiderivation $A : M \Rightarrow^* P$ with respect to another one $B : M \Rightarrow^* Q$ starting at the same term (in this situation A and B are said to be **coinitial**). The **residual** of A after B, in symbols $A\backslash B$, is the multiderivation $A\backslash B : Q \Rightarrow^* N$ that, intuitively, performs the steps of A that are left after performing B. Then A and B are **permutation equivalent**, in symbols $A \equiv B$, whenever $C\backslash A = C\backslash B$ for any multiderivation C coinitial with both. By defining the **join** $A \sqcup B$ of A and B as the concatenation $A; (B\backslash A)$, we can thus say that B performs more work than A if $B \equiv A \sqcup B$. The Parallel Moves Lemma, a strong form of the Church-Rosser property, states that $A \sqcup B \equiv B \sqcup A$, for coinitial multiderivations A, B. These ideas were introduced, for orthogonal term rewriting systems, by Huet and Levy (1979) who also developed the underlying theory: we summarize below the basic definitions and facts that we shall need, referring to this classic for the full details.

Given an elementary derivation $a : M \rightarrow_u N$ using the rewriting rule $\lambda = \rho$, and an arbitrary $v \in \mathcal{O}(M)$, the set $v\backslash a$ is the set of positions in $\mathcal{O}(N)$ defined as:

$$v\backslash a = \begin{cases} \{v\} & \text{if } v \mid u \text{ or } v \prec u \\ \{uw'v' \mid \rho/w' \text{ is a variable } x\} & \text{if } v = uwv' \text{ and } \lambda/w = x \\ \emptyset & \text{otherwise} \end{cases}$$

Observe that, if $w \in v - u$, then $M(w) = N(w)$. For a derivation $a; b$ obtained by concatenating the elementary derivation $a : M \rightarrow M'$ with the derivation $b : M' \rightarrow^* N$, and any $v \in \mathcal{O}(M)$, define

$$v\backslash a; b = \bigcup \{w\backslash b \mid w \in v\backslash a\}.$$

When $A : M \Rightarrow_U N$ is an elementary multiderivation with $U = \{u_1, \ldots, u_n\}$ and $v \in \mathcal{O}(M)$, define $v\backslash A = v\backslash(a_{\pi(1)}; \ldots; a_{\pi(n)})$, where $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ is any permutation and a_j is the elementary derivation contracting u_j. Then, for any multiderivation A and $v \in \mathcal{O}(M)$ define $v\backslash A$ by induction of the length $|A|$ of A as follows, where id is the empty multiderivation:

$$\begin{cases} v\backslash \text{id} & = \{v\} \\ v\backslash A; B & = \bigcup \{w\backslash B \mid w \in v\backslash A\}. \end{cases}$$

If $U \subseteq \mathcal{O}(M)$ consists of disjoint positions, then

$$U\backslash A = \bigcup \{u\backslash A \mid u \in U\}.$$

Finally, for coinitial multiderivations A, B with $B : M \Rightarrow_U N$, set

$$B\backslash A = U\backslash A,$$

and an application of the Parallel Moves Lemma justifies the extension of the residual relation to multiderivations of arbitrary length:

- For coinitial A, B, with B elementary multiderivation:

$$\begin{cases} \mathrm{id} \backslash B & = \mathrm{id} \\ (A_1; A_2)\backslash B = (A_1\backslash B); (A_2\backslash(B\backslash A_1)). \end{cases}$$

- For arbitrary coinitial multiderivations A, B:

$$\begin{cases} A\backslash \mathrm{id} & = A \\ A\backslash B_1; B_2 = (A\backslash B_1)\backslash B_2. \end{cases}$$

2.3 Outside-in reductions

In order to analyze the computational aspects of primitive recursive definitions relevant to the present discussion, we need to be able to put any derivation into a canonical, permutation equivalent form (Huet & Levy 1979). This generalizes the Standardization Theorem of the λ-calculus to orthogonal term rewriting systems, where the choice of the leftmost-outermost redex at each step does not lead in general to standard reductions.

For a redex occurrence $u \in \mathcal{R}(M)$ with redex scheme λ, we can define the set of positions that form the pattern of the redex at u as:

$$\mathcal{C}_M(u) = \{v \in \mathcal{O}(M) \mid u \preceq v \text{ and } \lambda/v \notin \mathcal{V}\}.$$

Given any position $u \in \mathcal{O}(M)$, say that a multiderivation $A : M \Rightarrow^* N$ **preserves** u if at each step of A no redex occurrence $v \succ u$ is contracted.

Definition 1 (Huet & Levy 1979, page 11). A position $u \in \mathcal{O}(M)$ is **external** for a multiderivation $A : M \Rightarrow^* N$, written $u \in \mathcal{X}(A)$, when

- either A preserves u, or
- A can be decomposed into $A_1; A_2; A_3$ such that:
  - A_1 preserves u,
  - $A_2 : P \Rightarrow_V Q$, $v \in V$ and $u \in \mathcal{C}_P(v)$, for some (unique) $v \prec u$,
  - $v \in \mathcal{X}(A_3)$.

Given a multiderivation

$$A : M_0 \Rightarrow_{U_1} M_1 \Rightarrow_{U_2} M_2 \Rightarrow_{U_3} \cdots \Rightarrow_{U_n} M_n$$

and $u \in \mathcal{R}(M_0)$, we say that u is an initial redex occurrence **pertaining** to A, written $u \in \mathcal{R}(A)$, if $U_i \cap u\backslash A[i-1] \neq \emptyset$ for some $i \leq n$, where

$$A[i-1] \overset{\text{def}}{=} M_0 \Rightarrow_{U_1} M_1 \Rightarrow_{U_2} M_2 \Rightarrow_{U_3} \cdots \Rightarrow_{U_{i-1}} M_{i-1}.$$

The set of **external redex occurrences** of A is defined as the set

$$\mathcal{E}(A) = \mathcal{R}(A) \cap \mathcal{X}(A).$$

A derivation A is **outside-in** if

- either $A =$ id, or
- $A = A_1; A_2$, for some elementary derivation A_1 contracting $u \in \mathcal{E}(A)$ and A_2 outside-in.

The main property of outside-in derivations is their ubiquity (Huet & Levy 1979, page 15): every multiderivation A can be transformed into an outside-in derivation B such that $A \equiv B$.

2.4 Computational behavior of primitive recursive definitions

Assume that a term M contains an occurrence u of a symbol defined by means of primitive recursion. For a derivation $a : M \to_v N$ contracting a redex M/v at v by applying the rule $\lambda = \rho$, define the relation

$$u \rhd_a w \qquad (u \textbf{ causes } w)$$

for $w \in \mathcal{O}(N)$ by the clauses:

- if $u = v$, then $u \rhd_a w$ for all positions $w \in \mathcal{O}(N)$ such that $w = uw'$, $w' \in \mathcal{O}(\rho)$ and $\rho(w')$ is a defined symbol;
- otherwise, $u \rhd_a w$ for all $w \in u\backslash a$.

Extend this notion of causality to elementary multiderivations $A : M \Rightarrow_U N$, by defining, for $u \in \mathcal{O}(M)$ and $w \in \mathcal{O}(N)$, $u \rhd_A w$ if and only if $u \rhd_a w$ for some elementary derivation a contracting $v \in U$: this definition is justified by the disjointness of redexes in U. Finally, for an arbitrary multiderivation

$$A = A_1; A_2 : M \Rightarrow_{U_1} M_1 \Rightarrow_{U_2} M_2 \Rightarrow_{U_3} \cdots \Rightarrow_{U_n} N$$

such that $A_1 : M \Rightarrow_{U_1} M_1$, $A_2 : M_1 \Rightarrow^* N$, with $u \in \mathcal{O}(M)$ and $w \in \mathcal{O}(N)$:

$$u \rhd_A w \text{ if and only if there is } u' \text{ such that } u \rhd_{A_1} u' \text{ and } u' \rhd_{A_2} w.$$

The **descendants** of the occurrence u of a defined symbol in M through a multiderivation $A : M \Rightarrow^* N$ are the elements of the set $\mathcal{A}_A(u) = \{w \in \mathcal{O}(N) \mid u \rhd_A w\}$. A **eliminates** the occurrence $u \in \mathcal{O}(M)$ of the defined symbol if $\mathcal{A}_A(u) = \emptyset$. Clearly, this notion is invariant under permutation equivalence.

A multiderivation

$$A : M = M_0 \Rightarrow_{U_0} M_1 \Rightarrow_{U_1} \cdots \Rightarrow_{U_{m-1}} M_m = N$$

is said to **contain** a multiderivation

$$B : N_0 \Rightarrow_{V_0} N_1 \Rightarrow_{V_1} \cdots \Rightarrow_{V_{m-1}} N_m$$

if each N_i is a subterm of M_i and the redexes in V_i, for $V_i \neq \emptyset$, have the form $u_i v$, where $N_i = M_i/u_i$ and $v \in \mathcal{R}(N_i)$. A multiderivation $A : M \Rightarrow^* N$, issued from a term of the form $M = M'[P]$, is said to **count up to** P if A contains multiderivations

$$B_1 : P \Rightarrow^* \mathsf{s}(P_1), B_2 : P_1 \Rightarrow^* \mathsf{s}(P_2), \ldots, B_{n-1} : P_{n-1} \Rightarrow^* \mathsf{s}(P_n), B_n : P_n \Rightarrow^* \mathbf{0}.$$

These give a multiderivation $B : P \Rightarrow^* \mathbf{n}$, where

$$\mathbf{n} = \underbrace{\mathsf{s}(\cdots\mathsf{s}}_{n \text{ times}}(\mathbf{0})\cdots),$$

for some $n \in \omega$. This notion is not in general preserved under permutation equivalence of derivations (consider for example the two equivalent derivations from the term $K(\mathbf{0}, C)$ using the rules $K(x, y) = x$ and $C = \mathbf{0}$), but we can still prove the following:

Lemma 2. *If $a : M[P] \to^* N$ is an outside-in derivation that counts up to P, then any $B \equiv a$ counts up to P.* □

Consider now the primitive recursive definition of addition:

$$+(x, \mathbf{0}) = x$$
$$+(x, \mathsf{s}(y)) = \mathsf{s}(+(x, y)).$$

We have the following property:

Proposition 3. *Any derivation A starting at $+(P, Q)$ which eliminates the root occurrence of the addition symbol counts up to Q.*

Proof: We can assume without loss of generality that

$$A : M_0 \to M_1 \to \cdots \to M_n$$

is already outside-in, and use induction on its length: the conclusion then follows by the preceding Lemma. If the position $\epsilon \in \mathcal{O}(+(P, Q))$ is not already a redex, it will eventually become one because A eliminates ϵ and therefore $\mathcal{A}_A(\epsilon) = \emptyset$, which implies that $\epsilon \backslash A = \emptyset$. Hence $A = A_1; A_2; A_3$ with $A_1 : M_0 \to^* M_i$, $A_2 : M_i \to_\epsilon M_{i+1}$ and $A_3 : M_{i+1} \to^* M_n$. As A is outside-in, all contractions in A_1 take place below position 2. In fact, assume that some position $1u$ is contracted in A_1 at step, say, M_k: then this position has to be external for the derivation

$$A[k, n] \overset{\text{def}}{=} M_k \to M_{k+1} \to \cdots \to M_n$$

and, not being preserved by A, this position must be necessary to create the redex at position ϵ in M_i, which is impossible as $\mathcal{C}_{M_i}(\epsilon) = \{\epsilon, 2\}$. Hence A_2 preserves 2, and this derivation is such that

1. either $A_1 : Q \to^* \mathsf{s}(Q')$, or
2. $A_1 : Q \to^* \mathbf{0}$.

In the second case, the root symbol is eliminated after performing $A_1; A_2$ and A counts up to Q. In the first case, we have:

$$A_1 : +(P, Q) \to^* +(P, \mathsf{s}(Q')),$$
$$A_2 : +(P, \mathsf{s}(Q')) \to_\epsilon \mathsf{s}(+(P, Q')),$$

and ϵ is external for A_3. Actually, in this case the position 1 is preserved by A_3, and we can consider an induced derivation:

$$A' : +(P, Q') = M'_0 \to M'_1 \to \cdots \to M'_n$$

eliminating the root symbol. A' is outside-in and $|A'| < |A|$, hence A' counts up to Q' by induction hypothesis, so that there is a derivation $Q' \to^* \mathsf{s}^{(n)}(\mathbf{0})$, for some $n \in \omega$. Then by composition we obtain

$$Q \to^* \mathsf{s}(Q') \to^* \mathsf{s}^{(n+1)}(\mathbf{0})$$

showing that A counts up to Q. □

A corresponding property can be shown to hold also for the primitive recursive definition of multiplication:

$$\begin{aligned} *(x, \mathbf{0}) &= \mathbf{0} \\ *(x, \mathsf{s}(y)) &= +(*(x,y), x). \end{aligned}$$

In this case we have:

Proposition 4. *Any derivation A starting at $*(P,Q)$ which eliminates the root occurrence of the multiplication symbol counts up to Q and, if $Q \to^* \mathsf{s}(Q')$ for some Q', then A counts also up to P.*

Proof: Assume, as before, that A is outside-in, and observe that ϵ must eventually become a redex occurrence contracted in the derivation $A[i,n]$, where

$$A : *(P,Q) = M_0 \to M_1 \to \cdots \to M_n.$$

Hence A can be decomposed into $A_1; A_2; A_3$, where

$$\begin{aligned} A_1 &: *(P,Q) \to^* M_i \\ A_2 &: M_i \to_\epsilon M_{i+1} \\ A_3 &: M_{i+1} \to^* M_n. \end{aligned}$$

As A is outside-in each redex contracted in A_1, say at step $k \leq n$, must be external for $A[k,n]$, hence they must be below the position 2 (in fact, by definition, the positions below 1 are not necessary for creating the redex at the root position in M_i). Then A_1 preserves 2, and we have that

1. either $Q \to^* \mathsf{s}(Q')$,
2. or $Q \to^* \mathbf{0}$.

In the second case A_1 counts up to Q (but not up to P). In the first case, after the contraction A_2 of the root redex, we have

$$A_3 : +(*(P,Q'), P) \to^* M_n$$

eliminating the descendants of ϵ in $A_1; A_2$. In particular, this entails that A_3 counts up to P by the preceding Proposition, hence A counts up to P. If $P \rightarrow^* \mathbf{s}^{(n)}(\mathbf{0})$ for some $n \in \omega$, then we can further decompose A_3 into

$$\begin{aligned} A_3^1 &: +(*(P, Q'), P) \rightarrow^* \overline{M} \\ A_3^2 &: \overline{M} \rightarrow^* M_n, \end{aligned}$$

where A_3^1 is the shortest initial segment of A_3 eliminating the root position of $+(*(P, Q'), P)$. We also know that no redex inside $*(P, Q')$ is contracted by A_3^1, hence

$$\overline{M} = \mathbf{s}^{(n)}(*(P, Q'))$$

and A_3^2 induces a derivation

$$A' : *(P, Q') \rightarrow^* M_n$$

that counts up to Q' by induction hypothesis, as $|A'| < |A|$. Finally, we have that A counts up to Q by suitably composing A_1 and the derivation showing that A' counts up to Q'. □

From this property it follows also that a derivation eliminating the root occurrence of a multiplication symbol from $*(P, Q)$ reduces that term to normal form, unlike what happens with addition.

Corollary 5. *If A starts at $*(P, Q)$ and eliminates the root occurrence of the multiplication symbol, let $Q \rightarrow^* \mathbf{m}$ and, if $m > 0$, let $P \rightarrow^* \mathbf{n}$. Then A counts up to $\mathbf{p}$, where $p = m * n$.* □

If we consider now the primitive recursive definition of exponentiation:

$$\begin{aligned} \uparrow(x, \mathbf{0}) &= \mathbf{s}(\mathbf{0}) \\ \uparrow(x, \mathbf{s}(y)) &= *(\uparrow(x, y), x). \end{aligned}$$

we see that, as a consequence of the preceding properties, in order to eliminate the root occurrence of the exponentiation symbol in $\uparrow(P, Q)$ it is necessary to count up to Q and, if $Q \rightarrow^* \mathbf{s}(Q')$ for some Q', also up to $\uparrow(P, Q')$ (and, by the preceding Corollary, up to $\uparrow(P, Q)$). There is, after all, a difference in the primitive recursive definitions of addition, multiplication and exponentiation that shows up in the form of the necessary conditions that the arguments and results of these functions have to satisfy in order to make possible the elimination of top level occurrences of the defined symbol. While in the cases of addition and multiplication we need only assume that some or all of the *inputs* are standard, the primitive recursive definition of exponentiation makes necessary to introduce the assumption that the (previous value of the) *output* is standard. This is one example in which a notation which is being introduced by a definition is assumed to be interchangeable with a primitive one, although this conclusion is not as strong as the kind of circularity revealed by the analysis carried out in (Isles 1992), because of the logic-free character of the equational formalism. Rather, the above results seem to enable a comparison with the typing of these functions in the 2-tiered version of the formalism proposed in (Leivant 1993b), having constructors $\mathbf{0}$ and $\mathrm{s}(\cdot)$ and tiers $\mathbf{N}_0, \mathbf{N}_1$, where a primitive recursive function defined by:

$$f(\vec{x}, \mathbf{0}) = g_0(\vec{x})$$
$$f(\vec{x}, \mathrm{s}(y)) = g_{\mathrm{S}}\left(f(\vec{x}, y), y, \vec{x}\right)$$

has tier $\mathcal{N} \times \mathbf{N}_i \to \mathbf{N}_j$ provided g_0, g_{S} have respectively tiers $\mathcal{N} \to \mathbf{N}_j$ and $\mathbf{N}_j \times \mathbf{N}_i \times \mathcal{N} \to \mathbf{N}_j$ for $i > j$, where $\mathbf{N}_i, \mathbf{N}_j \in \{\mathbf{N}_0, \mathbf{N}_1\}$ and $\mathcal{N}$ is a product of tiers of length $|\vec{x}|$. In fact we have that $+ : \mathbf{N}_0 \times \mathbf{N}_1 \to \mathbf{N}_0$ and $* : \mathbf{N}_1 \times \mathbf{N}_1 \to \mathbf{N}_0$, while exponentiation cannot be typed at all. It is tempting therefore to assign tier $\mathbf{N}_1$ to the arguments that will eventually be reduced to normal form (a unary numeral) in any computation eliminating the defined symbol.

2.5 A playful interpretation

Is there a formal sense in which the peculiar computational behavior of exponentiation involves some kind of circularity?

We imagine a Proponent of a primitive recursive definition who is committed to provide his Opponent with evidence that it can be eliminated. The scenario takes then the form of a dialogue between the Proponent and the Opponent, in the tradition of Lorenzen (1961) (see also (Stegmüller 1964) and the more recent (Felscher 1986)). We shall use these dialogical games as an heuristic device for understanding in a different way the assumptions behind the use of primitive recursive definitions in the above examples. Formal rules for them may be designed along the same lines as those for the dialogical games recently proposed in (Abramsky, Jagadeesan and Malacaria 1994) and (Hyland & Ong 1994) in their constructions of fully abstract models for typed λ-calculus with recursion, and from those for algorithms on sequential data structures described in (Curien 1992). (Actually, we shall mix the description of a typical play of the game with that of a "meta-game" between Proponent and Opponent, whose aim is to choose the rules in such a way that they are unbiased toward one of the players.)

A game is uniquely associated with a primitive recursive definition of the form:

$$F(x_1, \ldots, x_n, \mathbf{0}) = M[x_1, \ldots, x_n]$$
$$F(x_1, \ldots, x_n, \mathrm{s}(y)) = N[x_1, \ldots, x_n, y, F(x_1, \ldots, x_n, y)]$$

The positions are built by coloring the symbols occurring in primitive recursive terms by O or P: a player can only make moves relative to symbols of his color. Positions consist of **cells** or **values** where, for $\ell \in \{O, P\}$, ℓ-cells are filled by ℓ-values:

$$\text{cell} ::= O\text{-cell} \mid P\text{-cell}$$
$$O\text{-cell} ::= \mathrm{s}(O\text{-cell}) \mid {}^{O}F(\text{cell}, \ldots, \text{cell})$$
$$P\text{-cell} ::= \mathrm{s}(P\text{-cell}) \mid {}^{P}x \mid \text{timed } P\text{-cell}$$

$$\begin{aligned}
\text{timed } P\text{-cell} &::= {}^{P}x \mid \mathrm{s}(\text{timed } P\text{-cell})\\
\text{total } O\text{-value} &::= \mathbf{0} \mid \mathrm{s}(\text{total } O\text{-value}) \mid P\text{-cell}\\
\text{partial } O\text{-value} &::= \mathbf{0} \mid \mathrm{s}(O\text{-cell}) \mid \text{total } O\text{-value}\\
\text{total } P\text{-value} &::= \mathbf{0} \mid \mathrm{s}(\text{total } P\text{-value})\\
\text{partial } P\text{-value} &::= \mathbf{0} \mid \mathrm{s}(P\text{-cell}) \mid O\text{-cell}
\end{aligned}$$

Cells and values in these productions are mixed following very general principles of dialogical interaction that will be illustrated below in the context of examples. Later, in discussing how a play of the game for multiplication evolves according to such principles, we shall see an example of the possibility of dynamically transforming a cell into a value throughout a subgame in which an auxiliary question is answered.

The starting position of a game has always the form

$$^{O}F(^{P}x_1, \ldots, ^{P}x_n, ^{P}y)$$

with the Opponent to move. The labeling of symbols makes the input arguments available to the Proponent who may ask the Opponent to provide (piecewise) their values, while the Opponent asks for the (first piece of the) output value of the entire expression: the piecewise generation of values is intended to simulate the process of generation of numbers. Initially, however, the Proponent may require that some of the cells he owns be **timed**; the values with which timed cells are filled are total P-values $\mathrm{s}^n(0)$ written completely by the Opponent before the start of the play. Then the game proceeds by alternating O-moves and P-moves, that may be either **questions** or **answers**.

Let us consider for example the game associated to addition: the initial position is

$$^{O}{+}(^{P}x, ^{P}y),$$

where the Proponent has designed the cell ^{P}y as timed. Then the Opponent may make the move:

$$^{O}{+}(^{P}x, \mathrm{s}^n(\mathbf{0}))?$$

asking for the value of the cell labeled with his color. The Proponent has now either the possibility of asking for the value of the P-cell ^{P}x occurring inside the O-cell which he is trying to fill, or to use the primitive recursive definition of addition applied to the value of the timed P-cell ^{P}y, which yields the partial O-value $\mathrm{s}(^{O}{+}(^{P}x, \mathrm{s}^{n-1}(\mathbf{0}))$, a partial answer to Opponent's original question. Proponent is forced to choose the second alternative, as the first leads him to a situation in which he has no available move for answering Opponent's question. The game proceeds with Opponent to move from the position

$$\mathrm{s}(^{O}{+}(^{P}x, \mathrm{s}^{n-1}(\mathbf{0}))).$$

After a number of moves that is bounded by the value with which Opponent has filled the cell ^{P}y, the position in which Opponent is to move shall have the form

$$\mathrm{s}^n(^{O}{+}(^{P}x, \mathbf{0})).$$

The Proponent may now answer $\mathrm{s}^n({}^Px)$, by considering the cell $\mathrm{s}^n({}^Px)$ to be an O-value. The rationale behind this kind of move is provided by regarding a P-cell as a promise made by Opponent to fill it with a P-value. Using a rough analogy from economy, an ℓ-cell behaves very much like a promissory note or a bill of exchange issued by ℓ's partner and owned by ℓ. Clearly, then, it can be used by ℓ for discharging a debt to his partner. The play terminates, because Opponent is now unable to move further, hence this play is a win for the Proponent according to the usual termination convention for games of this kind. Actually, this play describes a winning strategy for the Proponent in an arbitrary game associated with addition: clearly the strategy is suggested by the proof of Proposition 3, including the choice of ${}^Py$ as a timed cell, for this is the argument up to which any derivation eliminating the addition symbol from the (term from which is built the) initial position of the game has to count.

In the case of multiplication, the initial position is

$$^O{*}({}^Px, {}^Py)$$

and the first move of the game can only be the Opponent's question:

$$^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^n(\mathbf{0}))?.$$

The Proponent, being unable to emit any value according to the primitive recursive definition of multiplication, may delay his answer by starting an auxiliary game with initial position

$$^O{+}({}^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^{n-1}(\mathbf{0})), \mathrm{s}^m(\mathbf{0}))$$

and Opponent to move. Observe that this move respects the constraints on admissible values for cells: being a partial P-value, the O-cell ${}^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^{n-1}(\mathbf{0}))$ can fill the P-cell at the first argument place, which is not timed. By this kind of move, the O-cell becomes **locked**, meaning that throughout the auxiliary game it is regarded as a value by the Opponent who cannot ask any question concerning this cell. This amounts, in terms of (real) dialogues, to Opponent making a *concession* to Proponent which has however a limited extent: as soon as the Proponent, in the develpment of this subgame, reaches an O-value, this is regarded as the answer to the question by Opponent that started the current subgame, the subgame terminates and the concession loses its validity by making the locked cell returning to behave as an O-cell. In the present example, after one move the Proponent produces the partial O-value $\mathrm{s}({}^O{+}({}^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^{n-1}(\mathbf{0})), \mathrm{s}^{m-1}(\mathbf{0})))$. After m such subgames the position becomes $\mathrm{s}^m({}^O{+}({}^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^{n-1}(\mathbf{0})), \mathbf{0}))$ with Proponent returning the O-value ${}^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^{n-1}(\mathbf{0}))$ which is then unlocked. The play continues from the position

$$\mathrm{s}^m({}^O{*}(\mathrm{s}^m(\mathbf{0}), \mathrm{s}^{n-1}(\mathbf{0})))$$

and eventually (that is, within a deadline established by Opponent as the total P-value n) the position becomes $\mathrm{s}^{mn}(\mathbf{0})$ with Opponent to move, a winning position for Proponent.

It is important that moves starting an auxiliary game be allowed only when an O-cell is substituted for a P-cell which is not timed. Otherwise the Opponent may refuse to accept the move on the basis of the fact that this is equivalent to accepting that the Proponent may delay his answer beyond any of the values that are accepted by the Opponent, and this clearly causes the rules to be biased towards the Proponent. This is instead exactly what happens in the case of exponentiation, in the position

$${}^{O}{*}({}^{O}{\uparrow}({}^{P}x, {}^{P}y'), {}^{P}x).$$

While the above argument outlines a winning strategy for Proponent in the game for multiplication, essentially as a consequence of Proposition 4, the initial question of Opponent cannot be answered by Proponent in the dialogue for exponentiation.

References

S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF (Extended Abstract). In Masami Hagiya and J.C. Mitchell, editors, *Theoretical Aspects of Computer Software, 1994*, Lecture Notes in Computer Science 789, pages 1–15. Springer-Verlag, Berlin-Heidelberg-New York, 1994.

S.J. Bellantoni. Predicative recursion and computational complexity. Technical Report TR 264/92, Department of Computer Science, University of Toronto, 1992.

S.J. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 1993.

P. Bernays. On Platonism in Mathematics. In P. Benacerraf and H. Putnam, editors, *Philosophy of Mathematics*, pages 274–288. Prentice-Hall, Englewood Cliffs, NJ, 1964. Originally appeared in French in *L'Enseignement Mathèmatique*, 1935, pp. 52–69.

É. Borel. *Les nombres inaccessibles.* Gauthier-Villars, Paris, 1952.

S. Buss. *Bounded Arithmetic.* Bibliopolis, Napoli, 1986.

P.-L. Curien. Concrete data structures, sequential algorithms, and linear logic. Message to the mailing list `types@theory.lcs.mit.edu`, June 3, 1992.

M. Davis. Why Gödel didn't have a Church's thesis. *Information and Control*, 54:3–24, 1982.

R. Dedekind. *Essays on the Theory of Numbers.* Dover, New York, 1963.

M. Dummett. Wittgenstein's Philosophy of Mathematics. *Philosophical Review*, 68, 1959. Reprinted in *Truth and Other Enigmas*, Duckworth, London, 1978, pp. 166–185.

M. Dummett. Wang's paradox. *Synthese*, 30:301–324, 1975. Reprinted in *Truth and Other Enigmas*, Duckworth, London, 1978, pp. 248–268.

M. Dummett. *Elements of Intuitionism.* Clarendon Press, Oxford, 1977.

J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

E. Engeler. An algorithmic model of strict finitism. In B. Dömölki and T. Gergely, editors, *Mathematical Logic in Computer Science*, Colloquia Mathematica Societas János Bolyai, 26, Amsterdam, 1981. North-Holland. This paper was written in 1971.

R.L. Epstein and W.A. Carnielli. *Computability: Computable Functions, Logic and the Foundations of Mathematics.* Wadsworth & Brooks/Cole, Pacific Grove, Ca, 1989.

A.S. Esenin-Vol'pin. Le programme ultra-intuitionniste des fondements des mathèmatiques. In *Infinitistic Methods*, pages 201–223. Pergamon Press-PWN, Oxford and Warsaw, 1961.

A.S. Esenin-Vol'pin. The ultra-intuitionistic criticism and the anti-traditional program for the foundations of mathematics. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 3–45. North-Holland, Amsterdam, 1970.

W. Felscher. Dialogues as a foundation for intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 3, pages 341–372. Reidel, Dordrecht, 1986.

R.O. Gandy. Church's Thesis and Principles for Mechanisms. In J. Barwise, H.J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pages 123–148, Amsterdam, 1980. North-Holland.

J.R. Geiser. A formalization of Essenin-Volpin's proof theoretical studies by means of non-standard analysis. *Journal of Symbolic Logic*, 39(1):81–87, 1974.

L. Henkin. On mathematical induction. *American Mathematical Monthly*, 67:323–338, 1960.

G. Huet and J.-J. Lévy. Call by need computations in non-ambiguous linear term rewriting systems. Rapport Laboria 359, IRIA, Aug. 1979.

J.M.E. Hyland and C.-H.L. Ong. On full abstraction for PCF. Draft, October 1994.

D. Isles. On the notion of standard non-isomorphic natural number series. In F. Richman, editor, *Constructive Mathematics*, Lecture Notes in Mathematics 873, pages 111–134. Springer-Verlag, Berlin-Heidelberg-New York, 1981.

D. Isles. What evidence is there that 2^{65536} is a natural number? *Notre Dame Journal of Formal Logic*, 33(4):465–480, 1992.

D. Isles. A finite analog to the Löwenheim-Skolem theorem. To appear in Studia Logica, 199?.

S.C. Kleene. *Introduction to Metamathematics.* Elsevier, New York, 1952.

J.W. Klop. Term rewriting systems. Technical Report CS R9073, Centrum voor Wiskunde en Informatica, Amsterdam, 1990.

D.E. Knuth. *The Art of Computer Programming*, volume 1. Addison Wesley, Reading, Ma, second edition, 1968.

G. Kreisel. Wittgenstein's Remarks on the Foundations of Mathematics. *British Journal for the Philosophy of Science*, 9:135–158, 1958.

D. Leivant. A foundational delineation of poly-time. *Information and Computation*, 1993.

D. Leivant. Stratified functional programs and computational complexity. In *Conference Records of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, New-York, 1993. ACM.

P. Lorenzen. Ein dialogisches Konstructivitätskriterium. In *Infinitistic Methods*, pages 193–200. Pergamon Press-PWN, Oxford and Warsaw, 1961.

G. Mannoury. Methodologisches und Philosophisches zur Elementar-Mathematik. Haarlem, 1909.

G. Mannoury. Woord en Gedachte. Groningen, 1931.

E. Nelson. *Predicative Arithmetic.* Princeton University Press, Princeton, 1986.

R. Parikh. Existence and feasibility in arithmetic. *Journal of Symbolic Logic*, 36(3):494–508, 1971.

C. Parsons. The impredicativity of induction. In M. Detlefsen, editor, *Proof, Logic and Formalization*, pages 139–161. Routledge, London and New York, 1992.

P.K. Rashevskii. On the dogma of natural numbers. *Russian Mathematical Surveys*, 28(4):143–148, 1973.

B. Rotman. *Ad Infinitum. The Ghost in Turing's Machine.* Stanford University Press, Stanford, CA, 1993.

V.Yu. Sazonov. On feasible numbers. *Journal of Symbolic Logic*, 57(1):331, 1992. Abstract of an unpublished paper with the same title.

J. Simon. On feasible numbers (preliminary version). In *Conference Record of the Ninth Annual Symposium on the Theory of Computing*, pages 195–207, New York, NY, 1977. Association for Computing Machinery.

A. Sochor. The Alternative Set Theory and its approach to Cantor's Set Theory. In H.J. Skala, S. Termini, and E. Trillas, editors, *Aspects of Vagueness*, Theory and Decision Library, pages 161–203, Dordrecht, 1984. Reidel.

W. Stegmüller. Remarks on the completeness of logical systems relative to the validity-concepts of P. Lorenzen and K. Lorenz. *Notre Dame Journal of Formal Logic*, 5(2):81–112, 1964.

A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936-37.

J. van Bendegem. Finite, empirical mathematics: outline of a model. Preprint 17, Rijksuniversiteit, Gent, Belgium, 1985.

D. van Dantzig. Is $10^{(10^{10})}$ a finite number? *Dialectica*, 19:273–277, 1956. Reprinted in Epstein & Carnielli 1989, pp. 258–261.

P. Vopěnka. *Mathematics in the Alternative Set Theory.* Teubner, Leipzig, 1979.

H. Wang. Eighty years of foundational studies. *Dialectica*, 12:466–497, 1958. All quotations are from the reprint of the paper in Hao Wang, Logic, Computers and Sets, Chelsea Publishing Company, New York, 1970, pp. 34–56.

L. Wittgenstein. *Bemerkungen über die Grundlagen der Mathematik.* Blackwell, Oxford, 1956. Edited by G.H. von Wright, R. Rhees and G.E.M. Anscombe.

L. Wittgenstein. *Philosophische Grammatik.* Blackwell, Oxford, 1969. Edited by R. Rhees.

C. Wright. *Wittgenstein on the Foundations of Mathematics.* Duckworth, London, 1980.

C. Wright. Strict finitism. *Synthese*, 51:203–282, 1982.

Logical Omniscience

Rohit Parikh[1]

Department of Computer Science
Broklyn College of CUNY and CUNY Graduate Center
33 West 42nd Streer, New York, NY 10036
email: ripbc@cunyvm.cuny.edu

In Plato's *Meno* [M] Socrates says at one point, "The man who does not know has within himself true opinions about the things that he does not know". This is said after a long conversation with one of Meno's attendants, an uneducated young boy, from whom Socrates elicits a particular version of Pythagoras' theorem solely by asking questions.

The question can then be raised, "Suppose you become convinced of some ϕ solely through being asked questions or through arguments based on facts that you already knew. Does it then follow that you already knew ϕ before you started?" This is an old puzzle about the role of argument. An argument is sound if the conclusion is already contained in the premises. But if so, then an argument can reveal nothing new and therefore someone who knows the premises already knows the conclusion.

The problem of logical omniscience revealed by this discussion reappears in the popular Kripke semantics for logics of knowledge. Under such semantics, a knower i has an accessibility relation $\approx_i$ which is usually an equivalence relation, but need not be assumed to be such for this discussion. Intuitively, $s \approx_i t$ means that the worlds s and t result in the same state of information for i. Then the formula $K_i(\phi)$ holds at a possible world (or state) s iff ϕ holds at all t accessible from s, i.e at all t such that $s \approx_i t$. It follows immediately that if ϕ is logically true then $K_i(\phi)$ holds at s, since it must hold at all such t. Also if $K_i(\phi)$ and $K_i(\phi \rightarrow \psi)$ hold at s then so does $K_i(\psi)$.[2] Thus what i knows at s includes all logical truths and is closed under logical consequence.

Given this state of affairs, i of course has no need of reasoning since i already knows everything that i might derive through it. But since real people (or processors) do not have these advantages, a more realistic theory of knowledge must allow for a knower not to know some logical truths or not to know some consequences of things that she knows. Recent literature has had a fair number

[1] Research supported by NSF grant CCR 92-08437

[2] For both ϕ and $\phi \rightarrow \psi$ must hold at all t accessible from s and hence ψ must hold at all such t. This yields the fact that $K_i(\psi)$ holds at s.

of papers addressed to this issue but the problem is still largely open.[3] Our purpose here is to survey some of the previous work and offer one or two suggestions.

1. Sentences and Propositions

One of the issues that turns out to be important is whether what is known is a sentence, i.e. a syntactic object in some language, or a proposition, which is the *sense* of such a sentence. The reason why this matters is that if ϕ and ψ are two sentences which are logically equivalent, then they will denote the same proposition and hence if it is propositions which are known, then knowledge of ϕ implies that of ψ. If on the other hand, it is sentences that are known, then someone who assents to ϕ but does not know of the equivalence may well dissent from ψ and hence we might want to say that he knows ϕ but not ψ. Thus for example, someone who asserts that his son is 2 years old might not want to say that his son's age is the only $n > 1$ for which there exist integers $x, y, z > 0$ with $x^n + y^n = z^n$.

Stalnaker [St91] also discusses the sentence-proposition issue, but decides in favour of propositions so that knowledge of sentences is not really discussed by him. He points out that with the choice of propositions as the objects of knowledge, if one knows ϕ and ϕ implies ψ, then one might not know ψ, but one must know $\phi \wedge \psi$, which is equivalent to ϕ.

The issue of the knowledge of sentences rather than propositions can also arise when there are complexity considerations. If we allow someone polytime algorithms to decide about the truth of something, then a long ψ might allow adequate time whereas a short and equivalent ϕ might not.[4]

This issue is addressed in [Pa87], one of the earlier papers devoted to logical omniscience. In the following definition, l stands for language and b for behaviour.

Definition [Pa87]: i has *external* knowledge of ϕ if ϕ is true in all situations which are compatible with i's evidence.

i has l-knowledge of ϕ if i has just said "yes" to the question whether ϕ is true and in all possible situations, i says "yes" to ϕ only if i has external knowledge of ϕ and says "no" to ϕ only when i has external knowledge that $\neg\phi$ holds.

i has b-knowledge of ϕ if there are three mutually incompatible actions α, β and γ such that i does α only if ϕ is true, does β only if ϕ is false, and moreover, i has just done α.[5]

Here l-knowledge does allow us to distinguish between knowing ϕ and knowing ψ even when the two sentences are equivalent. But for b-knowledge, if i

[3] One approach that has been tried is to use 'impossible' possible worlds. If i knows ϕ and also $\phi \rightarrow \psi$, but does not know ψ, then it is because there is an impossible, or illogical world from i's point of view where ϕ and $\phi \rightarrow \psi$ hold but ψ does not. This approach is discussed in [Li94] but we shall not discuss it here any further.

[4] A polynomial applied to a larger argument would allow more time than the same polynomial applied to a smaller argument.

[5] γ corresponds to "I don't know" so no conditions are imposed on it.

b-knows ϕ then he b-knows all equivalent ψ. A theromostat that turns on the heat when the termperature drops to 68°F will also turn on the heat when the teemperature drops to 20°C. This is also true of external knowledge, of which the Kripke semantics we described earlier is one special case, since given that ϕ and ψ are equivalent, evidence for one is also evidence for the other.

2. Implicit and Explicit Knowledge and the Question of Identity

If individual i knows ϕ and individual j knows $\phi \rightarrow \psi$, then together they know ψ, but neither might know ψ by itself. To bring out the knowledge of ψ from the implicit knowledge of it that the two possess, some communication between i, j must take place. What is interesting is that the question of logical omniscience even for a single individual is connected to this distinction.

Thus if a string x is written on the input tape of a Turing machine and the machine needs to know whether x belongs to some language L, the machine may need to carry out a computation. From the point of view of knowledge, this is puzzling and connected to the logical omniscience problem. For the machine knows that the string on the tape is x and of course it is a logical fact that this particular string *must* be in L, if it is. I.e., if s is a variable over strings and s_0 denotes the partcular string on the tape, en $s_0 = x$ is a fact that the machine knows, and $(\forall s)(s = x \rightarrow s \in L)$ is a logical truth. So why does the machine have to work to find out what it already knows?

But if we think of the string x as written over many squares of the tape and the head as the one who has to answer us, then the knowledge that $x \in L$ is implicit knowledge shared among all these *separate* individuals and naturally communication among these (i.e. computation) is needed to make the knowledge explicit.

This issue can be brought out in a very clear way by the following example. Suppose that i and j are processors connected by a one way channel from i to j. i knows ϕ and j knows $\phi \rightarrow \psi$. If we regard i, j as a single process $i + j$, then $i + j$ does know ψ, but to get this knowledge, one must ask at the imput port of i and wait for an answer at the output port of j. Thus while it is true that we can regard $i + j$ as a single individual which knows ψ, recovering this knowledge has certain side conditions.[6]

3. Computational Complexity

One approach to the question of logical omniscience is to take the help of complexity theory, which is also motivated by the desire to take resource limitations into account. One such approach is that of Moses [Mo88], where algorithms

[6] Indeed, when we ask a question over the telephone, we speak into the mouthpiece and expect our answer 'from' the earpiece

for deciding whether one knows ϕ are considered. Such an algorithm is *sound* if it only says that one knows ϕ when external knowledge of ϕ is present. Knowledge is now no longer closed under logical consequence. For example there is a polytime algorithm for deciding if a given graph is a simple cycle. Call this property ϕ. It is also always true that $\phi \rightarrow \psi$ where ψ is the property of having a Hamiltonian cycle. Hence of course there is a polytime algorithm for $\phi \rightarrow \psi$, namely the algorithm which always says "yes". However, assuming that $P \neq NP$, there is no polytime algorithm for ψ. Hence it is possible to poly-know ϕ, $\phi \rightarrow \psi$, but not ψ. However, one *can* polytime know $\phi \wedge \psi$ when there is polytime knowledge of ϕ, and $\phi \rightarrow \psi$ is valid.

Moses' approach is used by [HMT88] who analyse the zero knowledge protocols of Goldwasser, Micali and Rackoff and show that there is a precise technical sense in which zero knowledge protocols that prove that $x \in L$ show only *that* and no more. Any ϕ that is learned as a consequence of the dialogue is a consequence of $x \in L$. Their analysis requires us to take into account both computational complexity and probabilities, since these protocols yield not certain knowledge, but only knowledge with high probability.

4. BoundedRationality and Games

	C	D
C	3,3	5,2
D	2,5	4,4

Figure I

Figure I above is the problem known as the prisoner's dilemma. Two prisoners (row and column) are captured and questioned separately. Each has the choice (C) of co-operating with the other prisoner by not betraying him, and (D) of defecting, by testifying against him. The payoffs in terms of the number of years of imprisonment are given in each box so that if both co-operate, they each get a sentence of 3 years, but if they both defect (D), they each get a sentence of 4 years. Clearly, they are better off co-operating, but it is easy to see that for each of them, D is better than C regardless of what the other chooses. Thus they are both drawn to an outcome which is worse for both of them. Technically, this fact amounts to saying that there is a unique Nash equilibrium of (4,4) at the point (D,D) even though the outcome (3,3) at the point (C,C) is better for both.

One solution that has been suggested is the repeated prisoners' dilemma, where the same game is played repeatedly and one might expect that each prisoner would be motivated to co-operate since, if he betrays at some round, he would be afraid of retaliation at the next round. However, it turns out that the repeated prisoners' dilemma does not solve our difficulty. For if there are n rounds, then it is easy to see that neither prisoner has anything to lose by defecting at the n-th round. Since there is no next round, no retaliation is possible. Since defecting at the n-th round is the only rational thing to do, and they both know this, they might as well defect also at the $n-1$-th round, and backwards induction yields that they will defect in all rounds.

It turns out, however, that if the two prisoners (or rather the finite automata which model the prisoners in the paper [PY94]) are not *too* smart, then they will in fact do better than defecting (and thereby losing) in all rounds. [PY94] show that if the number of states of the finite automata is less than exponential, then there is a mixed Nash equilibrium[7] which yields outcomes arbitrarily close to (3,3). So stupidity in fact pays or can be made to pay.

It is interesting to point out that the use of finite automata to model limited reasoning powers is not at all new. Indeed this was one of the motivations behind the subject [RS59]. Both the pumping lemma and the Myhill-Nerode theorem are knowledge theorems in disguise. A finite automaton with n states can only know a finite amount ($log_2(n)$ bits) and hence reading a long string it cannot know how much of the string it has read. This yields the pumping lemma for regular sets. Similarly, given a finite automaton with n states, start state q_1 and transition function δ, there is an equivalence relation $\approx_M$ defined by $x \approx_m y$ iff $\delta(q_1, x) = \delta(q_1, y)$. The language L also defines an equivalence relation $\approx_L$ defined by $x \approx_L y$ iff $(\forall z)(xz \in L \leftrightarrow yz \in L)$. The finite automaton M with *some* accepting set F of states can accept L iff $\approx_M$ refines $\approx_L$. I.e. iff M can know at least as much about strings as membership in L requires.

5. Knowledge Algorithms

Ryle [R] insists that knowledge *how*, i.e. possesion of a skill, is primary and that knowledge *that*, e.g. knowledge of facts, presupposes and depends on knowledge how. The following definition is motivated by this observation.

Definition: [Pa87] A *knowledge algorithm* consists of a database together with a procedure that takes as input a question (say the truth value of some formula) and some resource bound, and operates on the question and the database upto some point determined by the value of the resource bound. Then either an answer is obtained or the bound is exceeded. In the first case, the answer is given and the database may be updated, even if the answer depended logically on evidence already at hand. In the second case the answer "I don't know" is given[8].

[7] A mixed Nash equilibrium is a probabilistic combination of pure strategies which for a one round game would be just C and D.

[8] This answer really should be distinguished from the same answer given when one knows that one lacks external knowledge.

The database may also be updated as a result of information received from the outside.

Such a definition does accomodate examples like those in *Meno.* Here is an ordinary dialogue.

q: Do you know the factorisation of 143?
r: Not off hand.
q: Is 11 a prime?
r: (After thinking a little) Yes.
q: Is 13 a prime?
r: Yes,
q: How much is 11 times 13?
r: Let us see; 11 times 10 is 110. 11 times 3 is 33. 110
plus 33 is 143. Oh, I see.
q: Can you factorise 143 now?
r: Of course, it is 11 times 13.

This dialogue above is a typical example of a situation which is a problem from the point of view of logical omniscience, but not at all a problem when we see it in terms of algorithms and updated databases. Here the initial database of r may have no primes or only small ones. However, after this particular dialogue, the primes 11 and 13 will be added and the question about 143 would of course be answered readily.

A very similar approach is adopted by [HMV94] who seem unaware of [Pa87]. They define an agent's local state to be a pair $< A, l >$ where A is an algorithm and l is the rest of his local state. In local state $< A, l >$ the agent computes whether he knows ϕ by applying the local algorithm to input (ϕ, l). Thus in both [Pa87] and in [HMV94] the emphasis has shifted from knowledge as fact to knowledge as procedure.

The puzzle of the muddy children has been much discussed in the literature. Some children having played in mud, have got their foreheads dirty. In fact, k of them have dirty foreheads. Every child can see everyone else's forehead, but not his/her own. The mother arrives on the scene and says "one of you has a dirty forehead." She then asks all the children one by one, if it knows its forehead is dirty. Strange to say, they *all* say, "I don't know."

In the conventional version of this puzzle, (see [HM84]) all the children are supposed to be perfect ($S5$) reasoners, and it can be shown that the kth dirty child does know that it has a dirty forehead, but with real children, this is only likely if k is one. Indeed, everyone knows that real children will behave differently from idealised children. The problem is to find a theory of real children.

To understand the puzzle properly, we should examine the conventional argument that the last dirty child uses to realise that its forehead is dirty. Let k be the number of dirty children. If $k = 1$, then the dirty child sees no one else who is dirty, and realises that it is itself the dirty child. Suppose now that we have justified the case $k = m$ and are considering the case $k = m + 1$. The $(m+1)$th dirty child sees m dirty faces and knows that $k = m$ if it is itself clean and $k = m + 1$, if it is dirty.

It then reasons: if $k = m$, then the child before me would be the last dirty child and would have realised, using the argument for $k = m$, that its forehead was dirty. However, it said, "I don't know." It follows that $k = m + 1$ and *my* forehead must be dirty.

What happens to this argument in a realistic setting?

Suppose that I am one of the children; all the children that I can see, who have muddy foreheads, have been asked if their foreheads are muddy and have already said "I don't know." I know that if my forehead was clean, then the dirty child before me should have said, "my forehead is dirty." Since that child said "I don't know," then *if* that child was capable of making that inference and did not, then it must be that my forehead is dirty.

However, what if my forehead is in fact clean, and the failure of the child just before me to realise that his forehead is dirty, is merely due to a weak reasoning capacity? In that case I should not assert that my forehead is dirty. Thus my being able to make the right inference about my forehead depends on my trusting the reasoning ability of the child before me, and his depends in turn on his trusting the children before him. Also, perhaps he said "I don't know if my forehead is dirty," not because he lacked the capacity to make the right inference, but doubted the capacity of the child before him. So the logical approach not only requires each child to be a perfect reasoner, but requires each child to assume that others are too.

Suppose, however, that I instruct the children: "I am going to put mud on some of your foreheads, and plain water on others. I will put mud on at least one forehead. Then I will point at each of you, one by one. If you do not see any child with mud on his forehead that I have not pointed to, and no one has yet raised his hand, then you must raise your hand."

This procedure can be learned by the children much more easily than learning any logic of knowledge. Moreover, if the children follow the instructions properly, then it will *always* be the case that the child who raises its hand will be the last muddy child. The earlier proof that the last dirty child knows that its forehead is dirty can be readily converted into a proof that the child that raises its hand will always be the last dirty child.

[Pa91] discusses other examples of situations where reasoning is replaced by some sort of 'unreflective' strategy and it is shown that the strategy that the logical argument gives is in fact the one that converges the fastest. However, even the fast strategy does not require reflection, only 'right action'. Moreover it is shown that there are also cases where a strategy that yields a high probability of being right converges much faster than one that delivers absolute knowledge.

In conclusion, knowledge is best seen as only an intermediary to action, and probably not all successful actions can be analyzed in terms of knowledge.[9] That said, of course the notion of knowledge is very useful and can often help us to organize the way we look at some situation.

[9] For exmaple, it is not helpful to say that someone who knows how to ride a bicycle must know some algorithm for doing this and is performing such an algorithm.

References

[CM86] M. Chandy and J. Misra, "How processes learn", *Distributed Computing* **1** (1986) pp. 40-52.

[Hi62] J. Hintikka, *Knowledge and Belief*, Cornell University Press, 1962.

[HM84] J. Halpern and Y. Moses, "Knowledge and Common Knowledge in a distributed Environment", *ACM-PODC 1984*, pp. 50-6l.

[Mo88] Y. Moses, "Resource-bounded knowledge" in *Theoretical Aspects of Reasoning about Knowledge*, ed. M. Vardi, Morgan Kaufmann 1988, pp. 261-276.

[HMV88] J. Halpern, Y. Moses, and M. Tuttle, "A Knowledge-based analysis of zero knowledge", in *Proc. 20th Annual ACM Symp. on Theory of Computing* pp. 132-147.

[HMV94] J. Halpern, Y. Moses, and M. Vardi, "Algorithmic knowledge", in *Theoretical Aspects of Reasoning about Knowledge*, ed. R. Fagin, Morgan Kaufmann 1994, pp. 255-266.

[Li94] B. Lipman, "An Axiomatic approach to the logical omniscience problem", in *Theoretical Aspects of Reasoning about Knowledge*, ed. R. Fagin, Morgan Kaufmann 1994, pp. 182-196.

[M] G. Grube (translator), *Five Dialogues of Plato*, Hackett publishing company, 1981.

[Pa87] R. Parikh, "Knowledge and the problem of logical omniscience" *ISMIS-87* (International Symp. on Methodology for Intelligent Systems), ed. Z. Ras and M. Zemankova, North Holland (1987) pp. 432-439.

[Pa91] R. Parikh, "Finite and Infinite Dialogues", in the *Proceedings of a Workshop on Logic from Computer Science*, ed. Moschovakis, MSRI publications, Springer 1991 pp. 481-498.

[PR85] R. Parikh and R. Ramanujam "Distributed Processing and the Logic of Knowledge", in *Logics of Programs*, Proceedings of a Conference at Brooklyn College, June 1985, Springer Lecture Notes in Computer Science #193. pp. 256-268.

[PY94] C. Papadimitriou and M. Yannakakis, "On Complexity as bounded rationality", in *Proc. 26th Annual ACM Symp. on Theory of Computing* (1994) pp. 726-732.

[RS59] M. Rabin and D. Scott, "Finite automata and their decision problems", *IBM Jour. Res. Dev.* **3** (1959) 114-125.

[Ry] G. Ryle, *The Concept of Mind*, Barnes and Noble 1949.

[St91] R. Stalnaker, "The Problem of logical omniscience" *Synthese* **89** (1991) pp. 425-440.

On Feasible Numbers*

Vladimir Yu. Sazonov

Program Systems Institute of Russian Academy of Sciences,
Pereslavl-Zalessky, 152140, Russia.
e-mail: sazonov@logic.botik.yaroslavl.su

Abstract. A formal approach to *feasible numbers*, as well as to *middle* and *small* numbers, is introduced, based on ideas of Parikh (1971) and improving his formalization. The "vague" set F of feasible numbers intuitively satisfies the axioms $0 \in F$, $F + 1 \subseteq F$ and $2^{1000} \notin F$, where the latter is stronger than a condition considered by Parikh, and seems to be treated rigorously here for the first time. Our technical considerations, though quite simple, have some unusual consequences. A discussion of methodological questions and of relevance to the foundations of mathematics and of computer science is an essential part of the paper.

1 Introduction

How to formalize the intuitive notion of *feasible numbers*? To see what feasible numbers are, let us start by counting: 0,1,2,3, and so on. At this point, A.S. Yesenin-Volpin (in his "Analysis of potential feasibility", 1959) asks: *"What does this 'and so on' mean?" "Up to what extent 'and so on'?"* And he answers: *"Up to exhaustion!"* Note that by cosmological constraints exhaustion must occur somewhat before, say, 2^{1000}, which is larger than the number of electrons in the universe! In a stricter sense, 2^{100} might also be viewed as non-feasible, but $2^{10} = 1024$ is surely feasible. The problem is that we cannot imagine any universally accepted border point between feasible and non-feasible numbers, seemingly precluding a systematic mathematical study of feasibility. Our aim here is to show that, quite to the contrary, feasibility is a notion that can be captured and analyzed by precise mathematical means.

> Nevertheless, according to quite a different approach a formal border point between "feasible" and "non-feasible" may be postulated to exist. We just *reject the abstraction of potential feasibility* in another way. We could postulate the existence of some *resource bounds* which always appear in practice and should not be neglected, as usually, but *explicitly taken into our consideration*, say, as parameters. This leads us to the idea of a finite row of natural numbers with the largest number (symbolizing the incidental resource bound), which may be denoted like zero as $\Box$. It proves that recursion theory relativized to such a finite row of

* Supported by Russian Basic Research Foundation (project 93-011-16016). This paper is an extended version of the abstract (Sazonov 1992).

natural numbers is essentially the theory of polynomial-time computability (Sazonov 1980), (Gurevich 1983), (Immerman 1982), (Vardi 1982). Then we may consider the corresponding version of Peano Arithmetic in $\{0, 1, \ldots, \Box - 1, \Box\}$ with two constants 0 and $\Box$ and the ordinary induction schema, etc. (Sazonov 1980a, 1989). Moreover, it makes sense to fix the value of $\Box$ to be equal, say, to 8 (the case of the chess-board 8×8). However, in this paper we will treat the "negation" of the abstraction of potential feasibility somewhat differently, without postulating any maximal (feasible) natural number.

So, if we denote the "vague" or "fuzzy" set of feasible numbers as F then we should postulate that $0 \in F$, $F+1 \subseteq F$ and $2^{1000} \notin F$ according to our intuition. However, it seems that Traditional Mathematics (both classical and intuitionistic) does not allow considering such postulates as consistent ones. Also, the approach of A.S.Yesenin-Volpin (which is sometimes called "ultraintuitionism" or "ultrafinitism", "actualism" (Troelstra 1990)) being very suggestive, appears too informal.

Troelstra and van Dalen (1988) also wrote on feasibility notion the following. "Natural numbers are usually regarded as unproblematic from a constructive point of view; they correspond to very simple mental constructions: start thinking of an abstract unit, think of another unit distinct from the first one and consider the combination ("think them together"). The indefinite repetition of this process generates the collection N of natural numbers. It should be pointed out that already here an element of idealization enters. We regard 5, 1000 and $10^{10^{10}}$ as objects of "the same sort" though our mental picture in each of these cases is different: we can grasp "five" immediately as a collection of units, while on the other hand $10^{10^{10}}$ can only be handled *via* the notion of exponentiation; 1000 represent an intermediate case. Visualizing $10^{10^{10}}$ as a sequence of units is out of the question. Exponentiation as an always performable *operation* on the natural numbers involves a more abstract idea than is given with the generation of N. < ... > There are considerable obstacles to overcome for a coherent and systematic development of ultra-finitism, and in our opinion no satisfactory development exists at present." (p.5–6.) "... certainly we have much less difficulty managing the idealized concept of the natural numbers, even though it is highly sophisticated one." (p.832.) "On the other hand, intuitionistically we do accept that "in principle" we can view $10^{10^{10}}$ as a sequence of units (i.e. we reject the ultrafinitist objection), and the authors are not sure that this is really less serious than the platonist extrapolation. At least it needs arguments." (p.851.) Also Borel (1947) mentioned that "the very large finite offers the same difficulties as the infinite".

Another informal consideration of feasibility notion was given in a popular lecture of A.N.Kolmogorov (1979) whose idea of *middle* and *small* numbers is formally developed in this paper.

A rigorous mathematical approach to feasibility was suggested by R.Parikh (1971) and developed further by other authors (V.P.Orevkov (1979), R.O.Gandy (1982), A.G.Dragalin (1985)). As Professor Gandy noted to the author, R.Parikh was the first who showed that feasibility indeed can be treated as mathematically coherent notion. However, the reason for writing the present paper is that his very interesting formalization of feasibility notion appears not to be completely adequate.

R.Parikh considered the ordinary Peano Arithmetic PA (in the language of primitive recursive functions) augmented with a new unary predicate F (which should not occur in the Induction Schema of PA!) and new axioms like the following:

$$0 \in F, \quad 1 \in F, \quad (x \in F \& y \in F \Rightarrow x + y \in F \& x \cdot y \in F \& \forall z \leq x (z \in F)),$$

and, most important,

$$2_{2^{1000}} \notin F.$$

Here it is defined by primitive recursion $2_0 := 1$ and $2_{k+1} := 2^{2_k}$ (and more generally, $2^x_0 := x$, $2^x_{k+1} := 2^{2^x_k} = 2^{2^x}_k$; $2^x_k = 2^{\cdot^{\cdot^{2^x}}}$, k times "2"). So, $2_{2^{1000}}$ denotes a huge value of exponential tower of 2^{1000} number of stages. The resulting theory PA$_F$ was proved in (Parikh 1971) to be practically (or feasibly) consistent in the sense that every formal proof of a contradiction in this theory should contain at least 2^{1000} symbols.

> More exactly, it follows from Parikh's theorems 2.2a and 2.2b (Parikh 1971) that in any tree-like Hilbert-style proof in PA$_F$ of the contradiction $0 = 1$ the number of logical axioms $A(t) \Rightarrow \exists x A(x)$ containing F or their quantifier complexity or the number of new F-axioms involved should be $> 2^{990}$.

This metamathematical statement is proved in the ordinary mathematical manner (roughly speaking, in the framework of Zermelo-Frenkel set theory or the like) and is based on Hilbert and Ackermann's ϵ-symbol elimination technique. In fact, R.Parikh and other authors (using also the cut-elimination technique) are concerned rather with obtaining complexity estimates for some proof parameters. However, we prefer to stress on reasonable concrete values of these parameters. We believe that, e.g. 2^{990} or even 2^{100} are non-feasible numbers and 1000 is feasible one in some *absolute* sense. So, their feasibility/non-feasibility does not depend on any computer technology. (Otherwise the parametric approach would be indeed the most reasonable.) That is why we will often use "finite"/"infinite" instead of "feasible"/"non-feasible". We also call such numbers as 2^{1000} as "imaginary finite" or simply "imaginary" or even as "infinite".

The number $2_{2^{1000}}$ is too rough upper bound for feasible numbers and without essential changes of the above approach we cannot replace such upper bound in the last axiom of R.Parikh by 2^{1000} or even by 2_{1000} (where $2^{1000} \ll 2_{1000} \ll 2_{2^{1000}}$). In fact, we can argue (by using the material of the next section) that provability of $2_{1000} \in F$ is in some exact sense inevitable here. So, even the intuitively true axiom $2_{1000} \notin F$ would be *contradictory* in PA$_F$. This means that Parikh's upper bound $2_{2^{1000}}$ for feasible numbers was sufficiently exact for the concrete formalism he used. Simultaneously, this witnesses that his theory (together with its underlying logic) is not completely adequate as a theory of feasible numbers. It is rather a first satisfactory approximation.

This paper is devoted to perform some further step to overcome this difficulty. It consists in finding suitable restriction on the underlying predicate calculus considered as Logic of Mathematics and in arguing that this restriction (probably[2]) does not crucially destroy our ability to develop mathematical knowledge. Moreover, this allows to consider even the feasible number 1000 as infinite in a suitable sense (cf. Vopenka's notion of "witnessed universe" (Vopenka 1979)). This restriction on logic (in its strongest form) proves to be quite simple. Its main clause has been well known for a long time but was not considered immediately in connection with feasible numbers. It consists just in *rejecting the cut rule* or, equivalently, in *allowing only normal natural deductions* in developing Mathematics. The basic aim of this paper is to demonstrate the adequacy of such a restriction. We also suggest some other more liberal and still adequate restriction.

Note, that there is a more rough approach to feasibility which also was initiated by R. Parikh (1971). Here only $\exists n(2^n = \infty)$ may be postulated, rather than $2^{1000} = \infty$. There were many corresponding works on Bounded Arithmetic where exponentiation is not a provably total function. In addition to abovementioned (Sazonov 1980a, 1989) we refer to (Buss 1986), (Nelson 1986) and to more recent books (Hájek and Pudlák 1993) and (Krajíček 1995) for further details and the literature.

The complexity theorists know very well that there is an essential difference between, say, binary and unary notation systems for natural numbers. So, the (imaginary) number 2^{1000} in binary notation has a quite feasible form $100\ldots0$ (only thousand zeros), but its unary representation $111\ldots11$ is non-feasible, which corresponds to our intuition about this number. That is why we prefer (not for practical aims) unary notation system which also properly reflects the counting process. It is very good that we also have binary, decimal and other number systems which allows to considerably abbreviate "unary" numbers. But this does not mean that e.g. each (feasible in length) binary string like $100\ldots0$ denotes some (feasible) number. Nonetheless, the tradition is so strong that even in Bounded Arithmetic the abbreviations of natural numbers are identified with the numbers themselves. This theory is Arithmetic only by the form of its axioms. Actually it proves to be a theory of binary strings. On the other hand, Peano and Primitive Recursive Arithmetic completely neglect any such distinctions because they are too strong and rough for this.

2 Why Consider Feasible Numbers?

Let us first ask the counter-question: *Why consider non-feasible numbers*? It seems that there is no need in Mathematics and in Applied Mathematics to spe-

[2] Note, that we consider this paper as a reassuring experiment, a reasonable step, but not as a final truth.

cially introduce them. Rather, such unrealistic things as non-feasible numbers or non-measurable sets, or the possibility "to make two apples from one which have the same form" (due to the Choice Axiom in Set Theory) etc. are *undesirable side effects* of various formal techniques. We prove by the Mathematical Induction that function 2^x is total. However, the computational practice shows that it is actually partial ($2^{1000} = \infty$)! This is an interesting and actually well known (in Science and in every-day life) strange effect when we immediately *see* something as "black" but nevertheless *think* (for a technical convenience, by a habit or for some other reasons) that it is "white".

Another principle which is postulated in Mathematics and Logic, despite its "false" consequences, is the transitivity of implication. For example, we may argue that the implication is not transitive in some real situations such as the following one: if somebody is a baby today then he will be a baby one month later, but after one hundred months he will be surely not a baby. Hence, one hundred applications of the transitivity of implication fail in this case.

It seems that the reason for such approaches to develop Mathematics which contradict to our ground intuition and experience is in neglecting the corresponding "vague" notions "feasible"/"non-feasible", "big"/"small", etc. as non-mathematical ones. Also the resulting ordinary working apparatus proves to be still extreemly successful, sufficient and adequate in many *other* important respects. However, why should we consider these traditional approaches as the best or the unique possible ones?

Note that the ordinary Complexity Theory also deals with feasibility problems. Therefore, for completeness' sake this important comparatively-quantitative approach to algorithms theory might be deliberately concerned also with *feasible numbers* (not just only with *feasible computations* of functions and predicates defined both on feasible and non-feasible arguments). Kolmogorov's Complexity Theory of Finite objects seems was sufficiently close to this idea. Probably its highest success in the *traditional* framework was the reason that this theory did not turn to feasibility considerations in a rigorous mathematical way.

Moreover, by the author's opinion feasible numbers could be the proper notion to set into the foundation of (Applied) Mathematics. This is simply another way to introduce complexity theoretic approach in Mathematics by reconsidering the initial fundamental notions. It seems that this could give more smooth connection of Mathematics with real computers. Also this is a different and hopefully more natural approach to so called "fuzzy" Mathematics as well as to (Feasibly) Constructive Mathematics. As is well known, the Ordinary Constructivism allows transformation of existence proofs to *potential* constructions of corresponding "existing" objects. In contrast, Feasible Constructivism should guarantee just feasible constructions of feasible objects.[3]

[3] There are also other approaches to Feasible Constructivism or, more precisely, to *Polynomial Constructivism* (Cook 1975), (Cook and Urquhart 1993), (Buss 1986), (Sazonov 1989) which are concerned with extracting from constructive existence proofs the corresponding polynomial-time constructions of possibly non-feasible (in the proper sense) finite objects.

Let us illustrate this on the following quasi-practical example. Consider a variation of the chess game which differs from the ordinary one essentially by allowing for whites and blacks to make just two moves of the pieces at once instead of one move. Let also the overall number of moves is bounded, say, by 100. Then we may easily prove that whites have a strategy which allows them at least not to lose the game. Indeed, otherwise they can move a knight forth-and-back after which blacks prove to be at the symmetrical position! Intuitively, it is clear that this proof of the existence of a strategy for whites is highly non-constructive. (For the ordinary chess game we have no proof at all!) On the other hand, from the point of view of the traditional constructivism we may potentially find (by successive trials) the required strategy for whites which may be considered even as a (huge) finite object of a bounded size. We believe that a Feasible Number Theory is a reasonable framework which will give a precise sense to the notion of Feasible Constructivity. Only then we could hope to prove the plausible hypothesis that there exists no feasibly-constructive proof for (the variation of or for the original) chess game that whites, say, have a winning or non-losing strategy.

If, nevertheless, whites do have an intuitively feasibly constructive strategy in any reasonable rigorous sense then this could be guaranteed just by a proof in a Feasibly Constructive Theory. Not only the ordinary Constructivism, but even Polynomial Constructivism mentioned in the above footnote can do nothing in this situation. It probably could work if we generalize 8×8 chess-board to $n \times n$ for sufficiently *large* n. However, what about 8×8?

3 Formal Systems Revised

What is (a proof in) a formal or an axiomatic system? It is necessary here to give a right answer to this question. The ordinary explanation of this notion is rather rough for our aim to formalize feasible numbers. We define a formal system e.g. as a finite set of rules $A_1 \dots A_k/A_{k+1}$ where A_i are some syntactically well formed (schemes of) formulas. However, we may consider finite (instances of the schematic) formulas A_i and their sequences (which are formal proofs according to the rules) in three ways:

1. as some *real* or *feasible* strings of symbols which may appear on a sheet of paper or in a computer memory,
2. as some *abstract*, imaginary finite strings which are considered only as *potentially-feasible*[4], or
3. *inside some mathematical (meta-) theory* (such as PA or ZF via a Gödel numbering or the like).

In the first two cases our intended subject is some axiomatized branch of Mathematics described by the given formal system and developed, respectively,

[4] As usual, this term does *not* mean something which *could be made feasible* in our sense. It rather allows to *think* about too long non-feasible strings which are considered as finite only by some sufficiently formal reasons or by an idealization.

in a *strictly* formal or *potentially* formal way. In the third case we are actually considering Metamathematics of the formal system, so making more precise the second case. (However, it may be asked, which way we prove metameta... mathematical results? Let us interrupt this infinite regress!) Of course, all these aspects are explicitly or implicitly involved in everyday mathematical activity. But the most essential point of this paper is that we should *not* mix them.

Let us consider for example the following first order theory T (regarded eventually as a finite list of formal axiom schemes and rules) of some weak arithmetic of natural numbers (even with *no* Induction Axiom at all). Non-logical symbols of T are one-place function symbol s for the successor operation $s(x) = x + 1$ and three-place predicate symbol R with the meaning $R(x, y, z) \Leftrightarrow x + 2^y = z$. There are only two special axioms in T recursively defining R:

$$T : \begin{cases} R(x, 0, sx) & (x + 2^0 = x + 1) \quad \text{and} \\ R(x, y, z)\&R(z, y, v) \Rightarrow R(x, sy, v) & (x + 2^{y+1} = (x + 2^y) + 2^y). \end{cases}$$

Note, that T does not prove that $x + 2^y$ is a total function, i.e. formally

$$T \nvdash \forall xy \exists z R(x, y, z).$$

Define the following sequence of formulas. $E_0(x) := x = x$; $E_{i+1}(x) := \exists y \in E_i.R(0, x, y)$ ($:= \exists y(E_i(y)\&R(0, x, y)) \Leftrightarrow$ "2^x is defined and $\in E_i$"). Hence $E_i(x)$ means that the value of 2^x_i is defined. Also take $N_0(x) := x = x$ and $N_{i+1}(x) := \forall y \in N_i \exists z \in N_i R(y, x, z)$ ($\Leftrightarrow \forall y \in N_i(y + 2^x \in N_i)$).

Theorem. $T \vdash E_{1000}(0)$ *(i.e.* $T \vdash$ *"*2_{1000} *is a finite number").*

Proof. (Essentially due to V.P.Orevkov (1979); cf. also R.Statman (1978, 1979).) We first infer $N_i(0)$, $i = 0, 1, 2, \ldots$, in T. For $i = 0, 1$ this is trivial. The case $N_{i+2}(0)$ is equivalent to proving the formula $\forall y \in N_{i+1}(y + 1 \in N_{i+1})$ or equivalently $\forall y[\forall x \in N_i(x + 2^y \in N_i) \Rightarrow \forall x \in N_i(x + 2^{y+1} \in N_i)]$. But the latter follows from the second axiom of T.

Then we can prove $N_i \subseteq E_i$ in T by induction on $i = 0, 1, 2, \ldots$. The case $i = 0$ is trivial. To prove $N_{i+1} \subseteq E_{i+1}$ we take any $y \in N_{i+1}$, i.e. any y such that (2^y is finite and) N_i is closed under addition of 2^y, and apply it to $0 \in N_i$. This gives $2^y = 0 + 2^y \in N_i \subseteq E_i$ and hence $y \in E_{i+1}$, as required.

It follows step-by-step that all $E_i(0), i = 0, 1, 2, \ldots, 1000$, are provable in T. □

We claim that there is something wrong in the above proof (which, however, seems very nice in itself). Indeed, what and how have been proved here? First of all, Mathematics and Metamathematics were mixed strongly. Even in the formulation of the theorem the expression $E_{1000}(0)$ is not a formula of our language but only a short denotation for some legitimate but rather long formula. ($E_i(x)$ contains exactly $3 + 13i$ symbols).

However, this is not the main difficulty with this proof. Of course, $E_{1000}(0)$ could be eventually written explicitly (13003 symbols are not so many). Much worse is the case with the (recursive) abbreviation N_{1000} which cannot be eliminated in practice because the intended formula of the original language evidently should contain $> 2^{1000}$ symbols. So, this direct attempt to make the proof rigorous i.e. to eliminate the Abstraction of Potential Feasibility, Metamathematics and other informal and illegal means does not succeed.

The conclusion is that the formula (denoted by) $E_{1000}(0)$ was not feasibly proved in T. We only proved that, potentially or metamathematically, there exists an *imaginary finite* proof of the formula $E_{1000}(0)$ in T. We cannot be completely satisfied by this metaproof of proof existence because we strongly believe that the *genuine* mathematical proof should be sufficiently short to be really written e.g. in a book.

A reasonable way of eliminating "meta" from metaproofs consists in extending the underlying predicate logic *to legitimate some formalism for required abbreviations.* This is a way to replace intuitive and metamathematical means by formal mathematical ones which we will adopt here[5]. Therefore we consider (some) abbreviation mechanisms, in general, as strong mathematical (rather than logical) tools.

Mathematical principles, in contrast to logical ones (as we understand them here), may have some special consequences about objects under consideration (e.g. the existence of very large numbers etc.). The above Theorem and especially considerations below show that some kinds of abbreviations indeed may have such consequences[6].

Of course, we cannot give the most general exhaustive mechanism for abbreviations to be used in Mathematics. This seems quite analogous to our inability to give a complete formalization of arithmetic or set theory. But we may introduce some useful and concrete such mechanisms[7]. Also we would not try to find abbreviations as strong as possible. Just as we choose some axioms and reject others in developing some branches of Mathematics, we will prefer only those most adequate abbreviation mechanisms which do not prevent us from formalizing the subject under consideration.

Remember that our present subject is feasible numbers. And the above theorem shows that theory T *together with all abbreviation mechanisms* used in it (both explicitly mentioned above and, even more important, implicit ones) is *non-adequate* for this aim: it proves that 2_{1000} is finite (what means here 'feasi-

[5] Another idea to introduce E_i and N_i, $i = 1 \ldots 1000$, immediately in the language of T and to consider their definitions as a new axioms of T seems not very appropriate. We must not change a *given* theory and its language depending on theorems to be proved. Let us play in a game with fixed rules!

[6] In fact, the abbreviations E_i and N_i in the above theorem prove to be not so crucial in this respect. We concentrated on them just to show the role of abbreviations in the general notion of formal proof. Yet more important for us is some other kind of abbreviations which were used in the above theorem *implicitly*. They are discussed below.

[7] Actually, in this paper we only mention such-and-such abbreviation mechanisms without introducing them rigorously. What we *will* do here, is putting a formal *veto* on some such mechanisms.

ble'). However, intuitively even $2^{1000} < 2_{1000}$ should be infinite! Note, that this consideration on theory T may be repeated for Parikh's formalization PA_F of feasible numbers mentioned in the Introduction above. I.e. there exists a feasibly long proof in PA_F (with some kind of abbreviations used, as above) of $2_{1000} \in F$. It follows that Parikh's upper bound $2_{2_{1000}} \notin F$ cannot be strengthened even to $2_{1000} \notin F$ without essential reconsidering the whole approach. Note that no general restriction on abbreviating means of PA_F was imposed in (Parikh 1971) (except those rather technical requirements on proofs which we mentioned in the Introduction).

Abbreviations may be applied not only to formulas, but also to proofs. For example, in the above theorem the proof of $N_{i+1} \subseteq E_{i+1}$ was described using *recursively* its subproof of $N_i \subseteq E_i$, and without this the resulting proof of $N_{1000} \subseteq E_{1000}$ should be rather long (instead of a proof occupying only a quarter of page).

However, *the most crucial abbreviations* widely used in Mathematics *deal with terms and objects* (in comparison with formulas and proofs as above). Let us consider e.g. the simplest term abbreviation $2{\cdot}x := x{+}x$. Then we may denote the number 2^{1000} as $2\cdot2\cdot2\cdot\ldots\cdot2\cdot1$ (thousand times '2'). This denotation, being rather long, is nevertheless quite feasible. However, it is impossible to really denote (even in a computer memory) such extraordinarily large number using only 1 and +, because this requires 2^{1000} occurrences of 1's. This suggests that *abbreviation of terms is not an appropriate tool if we want to formalize feasible numbers* (and thereby to exclude 2^{1000} from this numbers).

Note, that the ordinary formalizations of the predicate logic contain *implicitly* some kind of abbreviations of terms. In the case of the Natural Deduction Calculus (Prawitz 1965) we have the rules of introduction (I) and elimination (E) for each logical connective, for example for $\exists$:

$$\frac{A(t)}{\exists x A(x)}\,(\exists\mathrm{I}) \qquad \frac{\exists x A(x) \quad \begin{matrix}[A(x)]\\ \mathcal{D}\\ B\end{matrix}}{B}\,(\exists\mathrm{E})$$

where x is not free in B and in the open assumptions in $\mathcal{D}$, except $[A(x)]$, and quantification is understood up to proper renaming the quantified variable. So, the first rule *abbreviates* (possibly rather long[8]) *term t by the name x*. The second one *uses* the name x for some object satisfying A. If in a natural deduction some $\exists$-formula occurrence is both the conclusion of $\exists$I-rule and the main premise of $\exists$E-rule then x plays the role of an abbreviation of a term which is used in a deduction. We claim that these are such situations (likewise the abbreviation $2 \cdot x := x + x$) which give rise to non-feasible numbers and therefore they should be avoided. We will avoid analogous situations for other logical connectives as well: *introduced by an* I-*rule logical connective is not allowed to be eliminated by the corresponding* E-*rule.* The reason is that such subinference, for example,

[8] and even if not long, what about the iteration of such abbreviations?

$$\cfrac{\cfrac{A \quad B}{A\&B}\;(\&\mathrm{I})}{A}\;(\&\mathrm{E})$$

may lead to the above situation with existential quantification when the formula A is $\exists x\tilde{A}(x)$, its upper occurrence in the figure shown is the conclusion of some $\exists$I-rule and its lower occurrence is the main premise of $\exists$E-rule.

Slightly generalizing, this means that our restriction on proofs will consist in *allowing only normal natural deductions* (cf. the exact definition in (Prawitz 1965) and (Troelstra and van Dalen 1988)). In particular, this also means that we *can not freely use the general modus ponens rule* ($\Rightarrow$ E) with the corresponding rule ($\Rightarrow$ I)

$$\frac{A \quad A\Rightarrow B}{B}\;(\Rightarrow \mathrm{E}) \qquad \frac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A\Rightarrow B}\;(\Rightarrow \mathrm{I})$$

because the premise $A \Rightarrow B$ of the former could be introduced in the deduction by the latter so that the normality requirement fails (and again, as above, it may be considered the case when B is $\exists x\tilde{B}(x)$).

In this connection it is worth to remember an anticipating note in (Troelstra and van Dalen 1988): "The strictly finitist view also has its consequences for logic; the derivations of A and $A \Rightarrow B$ may still be within reach, but in order to apply modus ponens one might have to exceed the available natural numbers necessary for the length of the derivation of B." (p.29.)

Similarly, there is no guarantee that implication is transitive. This might seem very strange if we forget that our aim is to formalize such vague notion as feasible numbers or, say, Vopenka's notion of the *horizon* (Vopenka 1979) or the notion of *middle* or *intermediate numbers.* The latter notion could be considered as natural numbers counted before the horizon is "overcomed". Here $0,1,2,\ldots,10,11,\ldots$ are middle, 1000 and even 100 are definitely not middle (and lie "behind the horizon") and there is no maximal middle number. (Just look along a straight railway towards the horizon and count the pillars; see also the discussion in § 2 on non-transitivity of implication and § 4 below.)

Of course, every natural deduction can be (potentially!) normalized (Prawitz 1965). However, it is well known that this normalization (or cut elimination) process has *non-elementary lower (and upper) complexity bounds* (Orevkov 1979; Statman 1978, 1979). For example, consider the above proof of $E_{1000}(0)$ in the Natural Deduction form:

$$\cfrac{\begin{matrix}\vdots\\ N_{1000}(0)\end{matrix} \quad \cfrac{\begin{matrix}[N_{1000}(0)]\\ \vdots \\ E_{1000}(0)\end{matrix}}{N_{1000}(0)\Rightarrow E_{1000}(0)}\;(\Rightarrow \mathrm{I})}{E_{1000}(0)}\;(\Rightarrow \mathrm{E})$$

It is not normal[9] because the conclusion of ($\Rightarrow$ I)-rule is the main premise of ($\Rightarrow$ E)-rule. This proof cannot be normalized in practice. Indeed, every normal proof of the formula $E_{1000}(0)$ (which asserts the existence of the number 2_{1000}) will contain a term of the length $\geq 2_{1000}$ essentially due to the following form of

Herbrand's metatheorem. *Every normal classical deduction of $\exists$-formula $\exists\bar{z}C\bar{z}$ or $\neg\neg\exists$-formula $\neg\neg\exists\bar{z}C\bar{z}$ (possibly without $\exists$) from $\forall$- and $\neg\exists$-formulas of the kind $\forall\bar{x}A\bar{x}$ and $\neg\exists\bar{y}B\bar{y}$ can be reconstructed into quantifier-free normal deduction of some finite disjunction $\vee_i C\bar{t}_i$ from the formulas of the kind, respectively, $A\bar{t}$ and $\neg B\bar{s}$, and conversely. Moreover (*and most crucial*), in both directions the new deduction will contain only those terms which were occurring in the initial one.* □

Nevertheless, using abbreviations is extremely important tool of Mathematics. Therefore, it would be not very reasonable to reject all of them. That is why we summarize our *special requirements on mathematical proofs* by the following three rather informal clauses (except the clause 2):

1. **Arbitrary explicitly fixed abbreviation mechanisms for formulas and proofs, but not for terms are allowed.**
2. **Only normal proofs are allowed.**
3. **The number of symbols in a proof should be (intuitively) feasible.**

Of course, these requirements could be formulated more rigorously and also in a more strong or, on the contrary, in a more weak form. For example, in 3) 'feasible' could be strengthened by 'middle' or something such, because the genuine mathematical proofs should be *clear* and hence not only feasible, but also rather short.[10] However, it will be quite sufficient for the current aims and for simplicity sake to weaken our requirements as follows:

ONLY NORMAL PROOFS WITH FEASIBLE TERM SIZE ARE ALLOWED

Here *term size* of a proof is defined as the maximum of the number of symbols in each term occurrence in the proof. No restriction is imposed on the number of term occurrences and on the length of proofs and formulas in this final requirement. So, proofs and formulas, except terms, may be treated abstractly, as potentially feasible. This considerably simplifies the matter, and the resulting requirement on proofs naturally corresponds to the above three clauses. However, we believe that the ideal approach should be based on those clauses, probably with the technical rather strong restriction 2) replaced by some more liberal and hopefully more convenient one; cf. also § 5 below.

[9] and, in fact, uses abbreviations of terms via quantifier rules here not shown

[10] There are many examples when rather complicated mathematical proofs became very transparent and rather short after suitable reconsidering the presentation of the whole theory. After all, the main idea of any mathematically interesting proof is usually sufficiently simple.

Instead of the natural deduction we could formalize mathematical proofs by Gentzen's sequent calculus. In this case *the normal proofs may be equivalently replaced by cut-free ones.* However, we prefer natural deduction just because it is "natural" and our aim is, after all, to develop *Feasible Mathematics* with as minimum as possible extra technical efforts (which, however, are inevitable) connected with choosing any unnatural formalism for a real deducing theorems.

Let $\Gamma \vdash^n_f A$ mean that there exists some (possibly imaginary) *normal* classical first-order *natural deduction* (or *cut-free sequent deduction*) of A from Γ with intuitively *feasible term size.* To assert $\Gamma \vdash^n_f A$, it is sufficient to be able to really write down each term occurrence in the corresponding proof or even to be surely convinced that such a proof with short terms exists. $\Gamma \nvdash^n_f A$ will mean that there exists no such proof. We may be quite sure about this if, for example, we have some traditional meta-proof (in ZF or the like) that there is no required proof of the term size less than 2^{1000}. As an additional technical convention we will consider *the negation sign* $\neg$ *as definable one*:

$$\neg A := (A \Rightarrow \bot),$$

where $\bot$ is the primitive logical symbol denoting falsity with the ordinary *reductio ad absurdum* rule of inference (Prawitz 1965): $[\neg A] \ldots \bot / A$. In particular, we have the inference rule: $\bot$/everything. We will see below that this convention about negation plays an essential role in the consistency of a theory of feasible numbers FEAS defined in the next section.

It might be thought that the above notion of nf-proof is too vague due to the involved intuitive notion of feasible terms. However, the implicit use of the abstraction of potential feasibility of proofs in the ordinary approach to the notion of proof seems much more unclear. We believe that the only way of rigorous formalization of Mathematics is through feasible proofs (in various formal systems). On the contrary, the *potentially* feasible proofs in the *full* generality of this notion (if it has any sense at all) hardly can be considered as a genuine formal, i.e. mathematical one due to the "infinite regress" implicit in this notion.

> Note, that we are speaking here only about the formal and routine nature of mathematical proofs. How these proofs and corresponding formal systems are created is quite different question. Of course, this is usually extremely informal process. Nevertheless, any mathematical result should be presented in a sufficiently formal way. It is very important that the ordinary, not completely formal proofs in Mathematics usually can be transformed into feasibly formal ones (as the above proof of $E_{1000}(0)$) via explicating the necessary abbreviating mechanisms. The author's opinion is that it is this reason why any particular not very formal mathematical proof is actually considered by mathematical community as genuine mathematical one.

The reader might probably consider as a rather artificial the above normality requirement of proofs (taken simultaneously with the requirement on feasibility of term size). This also seems too strong restriction to the author himself. However note, that our aim was to find first *any* reasonable restriction which allows to formalize feasibility sufficiently adequately. On the other hand, we will present in § 5 some more liberal restriction on the predicate calculus which represents more directly our main idea that only terms must not be abbreviated. There may be also some other approaches and variations.

4 A Basic Theory of Feasible Numbers

The following theory, FEAS, is a point of departure for formalizing feasible numbers. The theory's non-logical symbols are $0, 1, +, \lfloor \log_2 \ldots \rfloor$ and $\leq$. Let FEAS_0 denote a collection of closed universal formulas (with terms of intuitively feasible length) which are feasibly true in an intuitive sense, such as $\forall x(x \neq x + 1 \neq 0)$, $\forall x \forall y \neq 0(x \leq \log_2 y \Rightarrow x + 1 \leq \log_2(y + y))$, $\log_2 1 = 0$ and, for definiteness, $\log_2 0 = 0$, i.e., several *ordinary* axioms. We will assume that this collection contains the feasibly true universal formulas that we need. Now define FEAS to be FEAS_0 extended with the *Main Axiom*

$$\forall y(\log_2 \log_2 y < \mathbf{10}),$$

that is, $2^{2^{10}} = 2^{1024} = \infty$; this too is a universal formula, and is intuitively *true for feasible numbers* (in a new natural sense respecting the old one). Those in doubt may wish to check this on a computer for various feasible y's represented in unary notation.

Now we assert the following facts about FEAS.

Fact 1. FEAS $\nvdash^{n}_{f} \perp$.

This assertion holds because any normal proof of $\perp$ in FEAS involves a term which is too long to be physically written down or stored, namely, we have

Metatheorem for FEAS. *Every normal proof of* $\perp$ *in* FEAS *contains a term with* $\geq 2^{1024}$ *symbols.*

Proof. By Herbrand's Theorem, in the form presented in § 3, each universal axiom of FEAS that occurs in a normal proof of $\perp$ can by replaced by closed instances thereof. The value of each closed term in the language of FEAS is bounded by its size. Substitution instances of axioms, including the Main Axiom, would be true in the standard sense if all substituted terms were of length $< 2^{1024}$. Therefore, true (in the standard sense) axioms would imply $\perp$, which is impossible. □

Another important corollary of Herbrand's Theorem is:

Fact 2. *The theory* FEAS *is a conservative extension of* FEAS_0 *with respect to closed quantifier-free and* $(\neg\neg)\exists$*-formulas. In fact, if a* $(\neg\neg\exists)$*-sentence has a proof in* FEAS *of term size* $< 2^{1024}$*, then it has a proof in* FEAS_0 *of the same term size.* □

This means that the two theories prove the same theorems about the terminaton of computations, that is, the kind of existential statements that is of greatest value in applications. Theorems of other forms, such as the Main Axiom $\forall y(\log_2 \log_2 y < \mathbf{10})$, are aimed at providing a reasonable abstract context for computations and algorithms (as in Hilbert's Program). Nonethelss, we have

Fact 3. *$\mathsf{FEAS} + \forall x(f(x) = x + x) \vdash^n_f \bot$, where f is a new function symbol for multiplication by two.* □

Thus the practically consistent theory FEAS becomes inconsistent once a name for the doubling function is introduced. This example have been discussed in § 3 above. Note that term size of the proof in Fact 3 is $\sim$ 1000 symbols. The details are left to the reader.

Let us define $M(x)$:= "x is a *middle* (or *intermediate*) *number*" := $\exists y \neq 0(x \leq \log_2 y)$ (here $y \neq 0$ is an inessential technical restriction to simplify one formal proof below) and $S(x)$:= "x is a *small number*" := $\exists y(x \leq \log_2 \log_2 y)$. Then, we have

Fact 4. *$\mathsf{FEAS} \vdash^n_f S(\mathbf{0})$, $\neg S(\mathbf{10})$, $\exists x(S(x)\&\neg S(x + 1))$, $M(\mathbf{0})$, $\neg M(\mathbf{1024})$, $\forall x(M(x) \Rightarrow M(x + 1))$, $\forall x \leq y(S(y) \Rightarrow S(x))$, $\forall x \leq y(M(y) \Rightarrow M(x))$ and $\forall x(S(x) \Rightarrow M(x))$.*

Note that provability of $M(\mathbf{0}), \neg M(\mathbf{1024})$ and $\forall x(M(x) \Rightarrow M(x + 1))$ gives no contradiction here. Indeed, the reader may see that the corresponding deduction

$$M(\mathbf{0}),\ M(\mathbf{0}) \Rightarrow M(\mathbf{1}),\ M(\mathbf{1}),\ M(\mathbf{1}) \Rightarrow M(\mathbf{2}),\ M(\mathbf{2}),\ \ldots,\ M(\mathbf{1024})$$

by multiple application of modus ponens rule is not normal one because $M(x) \Rightarrow M(x + 1)$ is actually deduced by introduction of implication rule (see the proof below), so modus ponens is not allowed. Of course, we could try to normalize successive subinferences of $M(\mathbf{1})$, $M(\mathbf{2})$, etc. by hand or by computer. However, this enterprise will be successful only for some initial part of this sequence. Surely, even $M(\mathbf{50})$ will be never "normally" proved (with any abbreviations for formulas and proofs but not for terms).

Proof of Fact 4. The cases of $S(\mathbf{0})$ and $M(\mathbf{0})$ are trivial. $\neg S(\mathbf{10})$ and $\neg M(\mathbf{1024})$ easily follow from the axiom $\forall y(\log_2 \log_2 y < \mathbf{10})$. The proof of $\forall x(M(x) \Rightarrow M(x + 1))$ uses the axiom on $\log_2$ by inferring first $M(x + 1)$ from $M(x)$:

$$\cfrac{M(x) := \exists y \neq \mathbf{0}(x \leq \log_2 y) \qquad \cfrac{\cfrac{[y \neq \mathbf{0}\&x \leq \log_2 y] \quad y \neq \mathbf{0}\&x \leq \log_2 y \Rightarrow x + 1 \leq \log_2(y + y)}{y + y \neq \mathbf{0}\&x + 1 \leq \log_2(y + y)}}{M(x + 1) := \exists z \neq \mathbf{0}(x + 1 \leq \log_2 z)}}{M(x + 1)}$$

Then $\forall x(M(x) \Rightarrow M(x+1))$ follows by introduction of implication and universal quantification.

Other cases are also easy except that of $\exists x(S(x)\&\neg S(x+1))$. To prove this we first write down the following natural deduction for several feasible numerals $\mathbf{n} = 0+1+1+\ldots+1$

$$\dfrac{\dfrac{[\neg\exists x(S(x)\&\neg S(x+1))]^4 \quad \dfrac{S(\mathbf{n}) \quad [\neg S(\mathbf{n+1})]^1}{\exists x(S(x)\&\neg S(x+1))}}{\perp}}{S(\mathbf{n+1})}\;1$$

Using successively these inferences for $\mathbf{n} = \mathbf{0}, \mathbf{1}, \ldots, \mathbf{8}$, together with the evident proof of $S(\mathbf{0})$, and the natural deduction

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\\ S(\mathbf{9})\end{matrix} \quad \dfrac{\dfrac{[S(\mathbf{10})]^3 \quad \dfrac{[\mathbf{10} \le \log_2\log_2 y]^2 \quad \dfrac{\forall y(\log_2\log_2 y < \mathbf{10})}{\log_2\log_2 y < \mathbf{10}}}{\perp}}{\perp}\;2}{\neg S(\mathbf{10})}\;3}{\exists x(S(x)\&\neg S(x+1))} \qquad [\neg\exists x(S(x)\&\neg S(x+1))]^4}{\perp}}{\exists x(S(x)\&\neg S(x+1))}\;4$$

we obtain the required normal proof with feasible term size. □

Analogously, FEAS $\vdash_f^n \exists x \in S \forall y \in S(y \le x)$. Note, that we can prove $\mathbf{0} \in S, \mathbf{1} \in S, \ldots, \mathbf{4} \in S$ where e.g. the last statement means that $2^{2^4} = 2^{16} < 10^5$ is a finite number. Much more difficult (if possible at all) it is to prove $\mathbf{5} \in S$ because $2^{2^5} = 2^{32} \sim 10^{10}$ is extremely large number. But $\mathbf{7}$ and even $\mathbf{6}$ are surely not in S because $2^{2^7} = 2^{128}$ and $2^{2^6} = 2^{64} = 16^{16} > 10^{16}$ are intuitively non-feasible (however less than 2^{1024}). It follows that intuitively the largest number in S is something like $\mathbf{4}$, or $\mathbf{5}$ and that $2^4 = 16 \in M$ but $2^6 = 64 \notin M$. These considerations also show that we could strengthen the main axiom of theory FEAS as $\forall y(\log_2\log_2 y < \mathbf{6})$ or even as $\forall y(\log_2\log_2 y < \mathbf{5})$. Such or other strong versions of feasibility axiom may depend on computer technology available today. However, we believe that the truth value (or, better to say, the role) of original feasibility axiom $\forall y(\log_2\log_2 y < \mathbf{10})$ does not depend on any technology.

We see that S and M have somewhat different properties. But the following fact shows that the story is much more intriguing. Simultaneously with the formula $\forall x(M(x) \Rightarrow M(x+1))$ we also have provable its negation:

Fact 5. FEAS $\vdash_f^n \neg\forall x(M(x) \Rightarrow M(x+1))$.

Proof. The following deduction of $\perp$ in the theory FEAS $+ \forall x(M(x) \Rightarrow M(x+1))$ is normal (despite many applications of modus ponens which are evidently allowed for partial cases of the *hypothesis* $M(x) \Rightarrow M(x+1)$):

> $\{M(\mathbf{0}), (M(\mathbf{0}) \Rightarrow M(\mathbf{1})), M(\mathbf{1}), \ldots, M(\mathbf{1023}), (M(\mathbf{1023}) \Rightarrow M(\mathbf{1024})),$ $M(\mathbf{1024}) := \exists y(\mathbf{1024} \le \log_2 y)$, so, let $\mathbf{1024} \le \log_2 y$, but this contradicts to the axiom $\forall y(\log_2\log_2 y < \mathbf{10})$, i.e. $\perp$ is proved$\}$.

Hence, the rule ($\Rightarrow$ I) gives FEAS $\vdash_f^n \neg\forall x(M(x) \Rightarrow M(x+1))$. □

We see that Facts 4 and 5 give FEAS $\vdash_f^n A$ and FEAS $\vdash_f^n \neg A$, for $A := \forall x(M(x) \Rightarrow M(x+1))$. But *this is not a contradiction* (cf. Fact 1). It follows only that the rule $A, \neg A/\bot$ is not always admissible, as well as the more general modus ponens (or cut) rule $A, A \Rightarrow B/B$ (because $\neg A$ is $A \Rightarrow \bot$). So, we should reconsider the question:

What is a Contradiction?

We adopt here the reasonable convention that a theory may be considered contradictory only if it is *trivial*, e.g. if all its well formed formulas (of feasible term size) are provable in our sense. Of course, FEAS is not such one. Nevertheless, the above unusual peculiarity of FEAS properly reflects the *contradictory nature* of such fuzzy notions as feasible and middle natural numbers. It is intuitively plausible that there exists no maximal middle number (e.g. the last month of our childhood) and, on the contrary, it is very strange to think that before 1000 there exists an "infinite" increasing sequence of natural numbers.

> The reader may remember also the related example of a picture on a computer display which looks simultaneously as *continuous and discrete*, and the reason for that is evidently just our mind, not an optical effect. The same holds for the physical continuum, because real numbers used in Physics have about 30 decimal digits after the point or, equivalently, about 100 (the intuitively non-middle number) of binary digits. So, there is the possibility to formalize real numbers as *infinite* sequences of binary digits so that each digit after the point will have the number *less* than 100 (or 1000). And the resulting continuum will be both continuous and discrete. Probably these finitary/infinitary ways of arguing could have some interesting effects if used simultaneously or mixed in one and the same proof. Also physical elementary particles which are considered both as particles and as waves may have some releavance to feasible numbers.

5 A More Liberal Approach

Let NK denote the classical calculus of natural deduction in the form of (Prawitz 1965). We will define below another restricted version NK^0 of the calculus NK whose inferences *do not use term abbreviations* in some more natural and not so restrictive sense than it was considered above. Deducibility in these calculi will be denoted, respectively, as $\Gamma \vdash^0 A$ and $\Gamma \vdash A$ where $\Gamma \vdash^0 A$ evidently must imply $\Gamma \vdash A$. The converse implication '$\Gamma \vdash A$ implies $\Gamma \vdash^0 A$' also will hold (classically), however, $\vdash^0$-deduction could be of too large size than the initial $\vdash$-deduction. Especially crucial for us is a possible lengthening the size of the participating terms due to using their abbreviations in the initial deduction.

Let, as above, the subscript ‘f’ denote deducibility with a (real) feasible size of participating terms. Then we will have trivially

$$\Gamma \vdash^0_f A \quad \text{implies} \quad \Gamma \vdash_f A$$

where the converse implication will *not always* hold. In particular, some theory T may be inconsistent in the sense of $\vdash_f$ ($T \vdash_f \bot$), but not in the sense of $\vdash^0_f$ ($T \nvdash^0_f \bot$).

On the other hand,

$$\Gamma \vdash (\vdash^0)\, A \quad \text{iff} \quad \Gamma \vdash^n A,$$

where ‘n’ symbolizes, as above, the *normality* property of natural deduction. The ‘if’ case holds because $\vdash^0$ will be defined to generalize $\vdash^n$. The ‘only if’ case may give rise to a considerable increasing the whole size of the deduction, *except* for the term size, as we will see below:

$$\Gamma \vdash^0_f A \quad \text{iff} \quad \Gamma \vdash^n_f A. \tag{1}$$

This is the reason why we could freely use more liberal notion of deducibility $\vdash^0_f$ instead of $\vdash^n_f$ for formalizing any feasibility theory, like FEAS. The calculus NK^0 or its ramified version NK$'$ defined below may be more comfortable to work with than normalized NK.

To define NK^0 let us consider an auxiliary calculus NK$'$ (with the corresponding deducibility relation $\vdash'$) obtained from NK by extending the first-order language of NK by a *weak* quantifiers $\forall'$ and $\exists'$ and by replacing NK-rules of *introduction* for $\forall$ and $\exists$ and also the *classical* NK-rule ($\bot_C$) for the falsity by the following rules

$$\frac{A(t)}{\exists' x A(x)}\,(\exists' \mathrm{I}) \qquad \frac{A(x)}{\forall' x A(x)}\,(\forall' \mathrm{I}) \qquad \frac{\begin{matrix}[\neg A]\\ \vdots \\ \bot\end{matrix}}{A}\,(\bot'_C)$$

where in the rule ($\bot'_C$) the formula A has not the form $\forall x B(x)$ or $\exists x B(x)$ (but, e.g., may have the form $\forall' x B(x)$ or $\exists' x B(x)$). In particular, quantifier *elimination rules* may be applied only to *strong* quantifiers $\forall$ and $\exists$

$$\frac{\exists x A(x) \qquad \begin{matrix}[A(x)]\\ \vdots \\ B\end{matrix}}{B}\,(\exists \mathrm{E}) \qquad \frac{\forall x A(x)}{A(t)}\,(\forall \mathrm{E}).$$

If $\mathcal{D}$ is an inference in the resulting calculus NK$'$ then $\mathcal{D}^0$ denotes the result of the replacement of all occurrences $\forall' x B(x)$ and $\exists' x B(x)$, respectively, by $\forall x B(x)$ and $\exists x B(x)$. Then we define NK^0-deductions as deductions of the form $\mathcal{D}^0$ where $\mathcal{D}$ is any deduction in NK$'$. We introduce analogously the denotations A^0 and Γ^0 for arbitrary formula A and list of formulas Γ in the extended language.

Note, that $\mathcal{D} : \Gamma \vdash' A$ implies $\mathcal{D}^0 : \Gamma^0 \vdash^0 A^0$. We also have that $\mathcal{D} : \Gamma \vdash^0 A$ iff $\mathcal{D}' : \Gamma' \vdash' A'$ for some $\mathcal{D}'$ such that $(\mathcal{D}')^0 = \mathcal{D}$, $(\Gamma')^0 = \Gamma$ and $(A')^0 = A$.

The calculus NK^0 explicates (via NK') the requirement that no quantifier introduced will be eliminated in a $\mathsf{NK}^0(\mathsf{NK}')$-derivation. Therefore, no (using of) term abbreviations are allowed in NK^0 and this is achieved with much more weak restriction than normality. However, we have the following connection with the approach presented before this section.

Detour-conversions (Troelstra and van Dalen 1988) (except for $\forall$- and $\exists$-*conversions* which are impossible here due to our splitting the quantifiers according to their I- and E-rules), $\bot_C$-*conversions*[11], *permutation conversions* and *immediate simplifications* are applicable to NK'-derivations as to NK-derivations. Moreover, no new terms will be introduced in the derivations during this process. It follows that (without using quantifier conversions) we may (potentially) normalize each NK'-derivation, as for the case of NK, however, evidently without introducing new terms and therefore with preserving the term size of the given NK'-derivation. This gives the one half of the above mentioned equivalence (1). The other half follows from a more general result that not only quantifiers, but also all logical connectives in a normal natural deduction may be split into the weak and strong ones according to their participating in the rules of the deduction. We postpone the corresponding detailed considerations to some other paper.

In the case of NK^0_f we have no usual semantics for logical connectives, e.g., for the implication $\Rightarrow$. So, modus ponens is again not always applied to $\Rightarrow$. The case of NK'_f seems better for the propositional connectives (no restrictions on the corresponding rules!). However the intuitive understanding the quantifiers requires some comments.

So, given a proof $\mathcal{D}(x)$ of $A(x)$ for an arbitrary x, i.e. essentially a proof of $\forall' x A(x)$, it may be problematic to obtain a proof of $A(t)$ for any term t of feasible size. The usual substitution $\mathcal{D}(t)$ works badly because t may have several occurrences in $\mathcal{D}(t)$ (or in some other term of $\mathcal{D}(t)$). One separate such substitution may be sufficiently harmless, and this could be considered as a justification of the missing in NK' rule ($\forall'$E). This makes corresponding strong and weak quantifiers $\forall$ and $\forall'$ "almost" the same. (Evidently, $\forall x A(x) \Rightarrow \forall' x A(x)$ and the converse implication was discussed just now as "almost" true.) However, doing this repeatedly may result in trying to consider deductions with terms of non-feasible size.

Analogous consideration is applicable to $\exists$ and $\exists'$ (as well as for the possible analogous splitting of implication or disjunction).

6 Further Possible Developments

To make the theory of feasible numbers more appropriate to applications in Computer Science it should be reformulated for more rich data types than nat-

[11] to be defined appropriately; in (Prawitz 1965) and (Troelstra and van Dalen 1988) there is no corresponding definition.

ural numbers, for example, for finite strings in some finite, e.g. binary alphabet or for hereditarily finite sets. Also intuitionistic as well as higher-order versions of theories discussed could be considered. A good mathematical theory should be sufficiently rich to describe a computability notion adequately. For example, in Bounded Set Theory (Sazonov 1987) provably recursive operations over HF-sets coincide with polynomial-time computable ones. Feasible Set Theory, if any, probably should have some features of BST and of Alternative Set Theory of P.Vopenka (1979).

In the case of strings, (feasible) natural numbers are identified with unary strings (those in one-letter-alphabet). Then, the addition operation "+" is generalized to the concatenation of feasible binary strings. Some other useful operations over strings may be introduced, as well, with the requirement: *The value of any* (intuitively) *feasible closed term should be a feasible string.*

Then binary strings of the length 1000 (or 100 or even 64, because intuitively $64 \notin M$) may be naturally considered as *real numbers* (in binary notation) between $0.000\ldots$ and $1.000\ldots = 0.111\ldots$. They are naturally factorised modulo equivalence relation of *approximate equality*

$x \approx y := \forall i \in M(x_i = y_i) \vee$ two symmetric disjuncts of the kind

$\exists j \in M[\forall i < j(x_i = y_i) \& x_j = 0 \& y_j = 1 \& \forall i \in M(i > j \Rightarrow x_i = 1 \& y_i = 0)]$

where x_i denotes the i-th digit of string x. We may try to develop Mathematical Analysis (e.g. to prove that $\sin' x \approx \cos x$, etc). The advantage of such a kind of Nonstandard Analysis would be corresponding "smooth" computability theory with real numbers containing only bounded (or even suitably fixed, e.g. 100) number of binary digits. Note, that, in contrast to the ordinary Robinson's approach, "nonstandard" methods seem inevitable in developing Feasible Mathematical Analysis.

The "set" of feasible numbers F may be naturally considered as a proper initial part of the set P of *polynomial numbers* (and strings) which is closed not only under addition "+" but also under multiplication "$\cdot$". It follows that $2^{1000} \in P \backslash F$, because $2^{1000} = 2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2$ and the right-hand side is now a legal term of feasible length. However, $2^{2^{1000}} \notin P$ because there is no feasible term in the language $0, 1, +, \cdot$ which denotes this "imaginary" number.

The corresponding theory POL of polynomial numbers *and* finite strings of polynomial length may be formulated (like FEAS) as POL_0 + the axiom

$$\forall y \in P(\log_2 \log_2 \log_2 y < \mathbf{10})$$

which means that $2^{2^{1024}} = \infty$. Here POL_0 denotes sufficiently reach list of closed $\forall$-formulas which are "true" over the "ordinary" finite binary strings. The language of POL and POL_0 consists of some finite list of symbols for functions over finite binary strings which are sufficient to express by terms all polynomial-time computable functions (cf. Sazonov 1980a, 1989). Moreover, it is required that this language does not contain the number function x^2. More exactly, we only have the multiplication $x \cdot y$ and, in particular, $x \cdot x$, where the abbreviation x^2

for the latter is not allowed and the corresponding one-place function symbol does not exist in the language. Otherwise, we could feasibly denote the (imaginary) number $2^{2^{1024}}$ as $(\ldots((2^2)^2)^2\ldots)^2$ and prove its finiteness and therefore infer the contradiction in POL.

We may define in POL feasible numbers by the predicate $F(x) := P(x)\&\exists y \in P(x \leq \log_2 y)$. Then middle and small numbers should be redefined, respectively, as $M(x) := P(x)\&\exists y \in P(x \leq \log_2 \log_2 y)$ and $S(x) := P(x)\&\exists y \in P(x \leq \log_2 \log_2 \log_2 y)$. (Cf. the definition of the predicates E_i in § 3.)

As in weak theories, say, of (Sazonov 1980a, 1989), the quantifier-free induction scheme is provable in POL. Also partial recursive functions via Turing computability can be described, as well as universal Turing machine, *s-m-n* theorem and recursion theorem (Sazonov 1980a, 1989). In particular, exponentiation 2^x is a *partial recursive function* here. It is undefined for $x =$ non-feasible polynomial number 2^{1024}, but its value on feasible number $x = 1024$ is defined and non-feasible. Then provably recursive functions of POL (i.e. those partial recursive functions whose totality can be proved in POL) are just polynomial-time computable ones (over polynomial binary strings).

Note, that theory FEAS, even if suitably extended from numbers to strings, seems not very appropriate to develop Turing computability. Indeed, multiplication missing in FEAS is necessary to estimate the time of simulating any Turing machine by an *universal* one and to prove this. Nevertheless we may try to define suitably the notion of partial recursive functions with the help of some other model of computation in such a theory. Then corresponding provably recursive functions could be naturally called *linear-time* or *feasibly computable.* Generally, such a way of arguing may be considered as a method to estimate "naturalness" of various notions of computability and complexity theory.

The discussion in § 3 shows that quantification (= term abbreviation) rules in non-normal proofs give rise to non-feasible (or to non-polynomial, etc.) numbers. However, if we know that the value of a (possibly very complicated) term is bounded by the value of some other term which will be never abbreviated in a proof, then the first term may be freely abbreviated without any such undesirable non-feasibility effect. Hence, the *normality requirement on proofs may be somewhat weakened as*:

> **Formula occurrences in a proof where normality fails (or cut formulas) must contain only bounded quantifiers. Additionally, Bounded Induction Axiom may be allowed.**

Here also something interesting may appear. Such abbreviations for finite binary strings of a bounded length, may give rise to not lengthy but very *complex strings* (cf. the notion of Kolmogorov's complexity of strings and the notion of constructive/non-constructive finite strings (Sazonov 1980a, 1989)). But we might want to consider only *simple* (i.e. not complex) *binary strings*, as it was above with feasible and polynomial numbers. So, we should choose which notion of binary string we are interested in and respectively decide whether the above kind of abbreviations for terms with bounded values is allowed or not.

It is very desirable to develop corresponding *informal* style of "Feasible Mathematics Thinking" like "Model-Theoretic Thinking" of classical mathematics which allows to prove theorems sufficiently rigorously, but without using formalized predicate calculus. For this aim we could begin with considering more and more liberal and convenient formalizations of corresponding logic as it was attempted above. After all, a good formalizm is such one which we use without too strong effort.

Note, that Gödel's argument on non-provability in arithmetic of its consistency does not work for formalisms (like FEAS) *without* modus ponens rule. Therefore, it is interesting to see what exactly will take place in our case.

The author sees no unsurmountable obstacle to proceed in these directions with all necessary technical details. However such a work evidently must be a heavy one because any restrictions on logic and on arithmetic require some more attention and ingenuity than usually.

7 Acknowledgements

The author is very thankful to N.V.Beljakin for numerous friendly and stimulating discussions on feasible numbers and foundational problems and to D.Leivant for the strong support, encouragement and for the polishing the English language of some parts of the paper.

References

1. Buss, S.R. (1986) *Bounded Arithmetic*, Bibliopolis, Napoli, 1986.
2. Buss, S.R. (1986) The polynomial hierarchy and intuitionistic bounded arithmetic, in: Structure in Complexity Theory, Lecture Notes in Computer Science **223** (Springer, Berlin) 125–143.
3. Cook, S.A. (1975) Feasibly constructive proofs and the propositional calculus, in: Proceedings 7th ACM Symposium on the Theory of Computation, 83–97.
4. Cook, S.A. and Urquhart, A. (1993) Functional interpretations of feasibly constructive arithmetic, Annals of Pure and Applied Logic **63**, 103–200.
5. Dragalin, A.G. (1985) Correctness of inconsistent theories with notions of feasibility, in: *Lecture Notes in Computer Science*, **208**, Springer-Verlag, 58–79.
6. Gandy, R.O. (1982) Limitations to mathematical knowledge, in: D. van Dalen, D.Laskar, J. Smiley eds., *Logic Colloquium'80*, North-Holland, Amsterdam, 129–146.
7. Gurevich, Y. (1983) Algebras of feasible functions, in: FOCS'83, pp.210–214.
8. Hájek, P., and Pudlák, P. (1993) Metamathematics of First-order Arithmetic, *Perspectives of Mathematical Logic*, 460 pp. Springer-Verlag.
9. Immerman, N. (1982) Relational queries computable in polynomial time, in: 14th STOC, pp. 147–152.
10. Kolmogorov, A.N.(1979) Automata and life (in Russian), *Kibernetika – neogranichennye vozmozhnosti i vozmozhnye ogranichenija. Itogi razvitija.* Moskwa, Nauka, 10–29.
11. Krajíček, J., (1995) Bounded Arithmetic, Propositional Logic and Complexity Theory, to appear in Cambridge University Press.

12. Nelson, E. (1986) *Predicative arithmetic*, Princeton University Press, Princeton, New Jersey.
13. Orevkov, V.P. (1979) The lower bounds of complexity the deductions increasing after cut elimination. In: *Zapiski nauchnych seminarov LOMI AN SSSR*, **88**, 137–162. (In Russian)
14. Parikh, R. (1971) Existence and feasibility in arithmetic, *JSL*, **36**, (3), 494–508.
15. Prawitz, D. (1965) *Natural Deduction*, Stockholm.
16. Sazonov, V.Yu. (1980) Polynomial computability and recursivity in finite domains. *Elektronische Informationsverarbeitung und Kybernetik*, **16**, (7), 319–323.
17. Sazonov, V.Yu. (1980a) A logical approach to the problem "$P = NP$?", in: *Lecture Notes in Computer Science*, **88**, Springer, New York, 562–575. (An important correction to this paper is given in [*Lecture Notes in Computer Science*, **118**, Springer, New York,1981, p.490.])
18. Sazonov, V.Yu. (1987) Bounded set theory and polynomial computability, FCT'87, *Lecture Notes in Computer Science*, **278**, p.391–397.
19. Sazonov, V.Yu. (1989) An equivalence between polynomial constructivity of Markov's principle and the equality $P = NP$ (in Russian), in: *Trudy instituta matematiki, Sibirskoje otdelenie akademii nauk SSSR, "Matematicheskaja logika i algoritmicheskije problemy"*, Novosibirsk, "Nauka", Sibirskoje otdelenije, 138–165. (See also shorter English version with the same title in P.Petkov ed., *Mathematical Logic*, Proceedings of the Heyting's conference, sept., 1988, Varna, Plenum Press, New York, 1990, 351–360.).
20. Sazonov, V.Yu. (1992) On feasible numbers, Abstracts of papers of European Summer Meeting of the Association for Symbolic Logic, Logic Colloquium'89, Berlin, *JSL*, **57** (1) 331.
21. Statman, R. (1978) Bounds for proof-search and speed-up of the predicate calculus. *Ann. Math. Logic.*, **15** (3), 225–287.
22. Statman, R. (1979) Lower bounds on Herbrand's theorem, *Proc. of the AMS*, **75** (1).
23. Troelstra, A.S. (1990) Remarks on intuitionism and the philosophy of mathematics (revised version), *ITLI Prepublication Series* X-90-01, University of Amsterdam, 18 pp.
24. Troelstra, A.S. and van Dalen, D. (1988) Constructivism in Mathematics. An introduction, Vol. I, II, North-Holland, Amsterdam.
25. Vardi, M.Y. (1982) The complexity of relational query languages, STOC'82, pp.137–146.
26. Vopenka, P. (1979) *Mathematics in the Alternative Set Theory*, Leipzig.
27. Yesenin-Volpin, A.S., (1959) Analysis of the potential feasibility, in: *Logicheskije issledovanija*, Moskwa, AN SSSR, 218–262. (In Russian).

On Parallel Hierarchies and R_k^i

Stephen Bloch

Math/Computer Science Dept, Adelphi University. *

Abstract. This paper defines natural hierarchies of function and relation classes, constructed from parallel complexity classes in a manner analogous to the polynomial-time hierarchy. A number of structural results about these classes are proven: relationships between them and the levels of PH, a Buss-style witnessing theorem relating the levels of these hierarchies to definability in the bounded arithmetic theories R_k^i (generalizing [1] and improving on [9]), a conservation result between S_k^i and R_k^{i+1}, and results analogous to those of [18, 8, 16] relating conservation between theories of bounded arithmetic to the collapse of complexity classes.

1 Introduction

Rohit Parikh [21] studied weak theories of arithmetic that have induction only for bounded, or Δ_0, formulæ. These theories became known as *theories of bounded arithmetic*, and were further developed by Sam Buss in [6]. Buss restricted induction not only to bounded formulæ, but to bounded formulæ of fixed complexity Σ_i^b. Furthermore, he distinguished between an exponential-length and a polynomial-length form of induction, defining thereby two related hierarchies of logical theories T_2^i and S_2^i, and proved basic relationships among them such as that $S_2^i \subseteq T_2^i \subseteq S_2^{i+1}$ for $i \geq 1$. Most importantly, Buss showed that the proof strength of S_2^i corresponded exactly to accepted notions of computational complexity: the functions at level i of the polynomial time hierarchy are precisely those with a Σ_i^b graph, provably total and single-valued in S_2^i.

Expanding on this work, various researchers (e.g. [1, 11, 12]) introduced a third hierarchy of theories R_2^i based on log-length induction, and demonstrated that R_2^1 analogously characterized the functions computable in the parallel class $\mathcal{NC}$. But no natural description was known of the computational content of R_2^i proofs for $i > 1$, in part because many natural arguments about log-length induction seem to require quasipolynomial, rather than polynomial, growth rates (although see [5] for an exception). Attention therefore turned to the theories R_3^i, in which terms have quasipolynomial growth. Buss, Krajíček, and Takeuti [9] described the problems for which R_3^i can prove the existence of solutions, relying on a somewhat complex notion of multi-valued functions computed with a limited number of queries to a witnessing oracle.

* This work was sponsored in part by Judy Goldsmith's NSERC operating grant OGP0121527 while the author was at the University of Manitoba; further work was done at the University of Kentucky.

The present paper characterizes (see Theorem 35) the functions provably total and *single-valued* in R^i_2, R^i_3, R^i_4, and so on, without the complications of multi-valued functions, witnessing oracles, or artificial bounds on the number of oracle queries. These simpler characterizations depend both on the function-theoretic characterization of [4] and on circuits with oracle gates, as in Wilson [25, 26]. But where Wilson constructed oracles at recursive and higher levels to prove relativized complexity results among classical complexity classes like $\mathcal{NP}$, the present paper uses oracles within classical complexity classes to construct and describe new classes analogous to those of the polynomial hierarchy.

We answer a question posed in [9] by proving that if $R^i_k \vdash (\forall x)(\exists y)\phi(x,y)$, for $\phi \in \Sigma^b_i$, then there is a function, provably total *and single-valued* in R^i_k, that witnesses the statement. (See Theorems 41 and 44.) We also give, in Theorem 47, a simpler proof of the conservation result in [9]. That paper also asked whether the known $\forall\exists\Sigma^b_{i+1}$ conservativity between S^i_3 and R^{i+1}_3 also holds between S^i_2 and R^{i+1}_2; we give circumstantial evidence that it does *not*, as such conservativity would imply a collapse of complexity classes (see Theorem 49).

Knowing that S^i_2 characterizes level i of the polynomial time hierarchy, or that R^i_k characterizes level i of a natural parallel hierarchy, does not in itself answer questions about the collapse or non-collapse of complexity classes. Krajíček, Pudlák, and Takeuti [18] showed that if $S^{i+1}_2 \equiv T^i_2$, then $\Sigma^p_{i+1} \subseteq \Delta^p_{i+1}$/poly and therefore the polynomial hierarchy collapses to at most $\Sigma^p_{i+2} = \Pi^p_{i+2}$. Buss [8] simplified the proof and strengthened the result: under the same hypotheses, $\Sigma^p_{i+1} \subseteq \Delta^p_{i+1}$/poly, and therefore the polynomial hierarchy collapses, *provably in* T^i_2. The present paper shows (see Theorem 51) that any equivalence of the form $R^{i+1}_k \equiv S^i_k$, with $i \geq 1, k \geq 3$, would imply a similar collapse of complexity classes, e.g. (for $k = 3$) the collapse of the quasipolynomial hierarchy, provably in S^i_k.

Another recent result, due to Krajíček [16], shows that if $S^i_2 \equiv T^i_2$, then $\mathcal{P}^{\Sigma^p_i} = \mathcal{P}^{\Sigma^p_i}[O(\log(n))] = \mathit{LogSpace}^{\Sigma^p_i}$. We have extended Krajíček's result in several ways: any equivalence $S^i_k \equiv T^i_k$ *or* $R^i_k \equiv S^i_k$, with $i \geq 1$, $k \geq 2$, would imply that the sequential function class $\Box^p_{i,k}$ can be parallelized to run exponentially faster; see Corollaries 34,45, and 46.

Section 2 defines models of parallel computation, and the complexity classes of the parallel hierarchies, used in this paper. Section 3 proves a variety of basic relationships among these classes. Section 4 shows a direct link between these classes and those defined in [9, 16]. In Sect. 5 is a Buss-style witnessing theorem to show that the classes $\Box^c_{i,k}$ correspond precisely to the Σ^b_i consequences of R^i_k; this implies an alternate proof of the link shown in Sect. 4, and an alternate proof of the conservativity result of [9]. Section 6 gives results analogous to those of [18, 8] relating the collapse of theories of bounded arithmetic to the collapse of complexity classes. We conclude by discussing some of the remaining results we would most like to prove about these theories.

The present paper is intentionally informal in places, omitting the least interesting proofs and definitions. A version with these details intact has been submitted for journal publication.

2 Definitions

2.1 Languages, Notation, and Growth Rates

We work with several first-order languages of arithmetic, similar to those used in [6, 9], with whose notation I assume the reader is familiar. First we define an hierarchy of term languages, essentially the same as that in [22].

Definition 1. The language $\mathcal{L}_1$ consists of

- the variable symbols a, b, c, d, w, x, y, z, possibly subscripted with an integer,
- the constant symbols 0 and 1,
- the binary infix relation symbols $=$ and $\leq$,
- the binary infix function symbols $+$ and $\cdot$, and
- the unary outfix function symbol $|\cdots|$,

Definition 2. For $k > 1$, the language $\mathcal{L}_k$ is the language $\mathcal{L}_1$ augmented with the binary infix function symbols $\#_2, \#_3, \#_4, \ldots \#_k$, pronounced "smash two," "smash three," and so on. "Smash two" is often simply called "smash".

These symbols are intended to be interpreted as follows: $x \#_2 y = 2^{|x| \cdot |y|} - 1$, $x \#_3 y = 2^{|x| \#_2 |y|} - 1, \ldots, x \#_k y = 2^{|x| \#_{k-1} |y|} - 1$. Smash functions are discussed in [13, 15, 20, 6, 1], all of which use unimportantly different definitions (for example, omitting the "-1").

We shall frequently refer to "the length of an $\mathcal{L}_k$-term," or $|\mathcal{L}_k|$ for short. For example, if we say "$f(x) \in |\mathcal{L}_1|$" or "$f(x)$ is bounded by the length of an $\mathcal{L}_1$ term," we mean that f is at most a linear function in the length of x. Similarly, $|\mathcal{L}_2|$ means a polynomial, or $2^{O(\log)}$, in the lengths of the free variables; $|\mathcal{L}_3|$ is quasipolynomial, or $2^{2^{O(\log \log)}}$, in the lengths of the free variables, and so on. We shall likewise refer to "the log of the length of an $\mathcal{L}_k$-term," abbreviating it $||\mathcal{L}_k||$ (the slight abuse of notation, equating log with length, doesn't affect the results).

2.2 Syntactic Complexity

We next define an hierarchy of syntactic formula classes, analogous to the classes Σ_i, Π_i, Δ_i familiar to recursion theorists. The following definitions differ from Buss's in [6] mainly in the presence of a second subscript k; since Buss was only concerned with a language analogous to $\mathcal{L}_2$, his results apply directly to our classes with $k = 2$.

Definition 3. A bounded quantifier with bound of the form $|t(x)|$, i.e. one whose outermost function symbol is $|\cdots|$, is said to be *sharply bounded.*

Definition 4. The classes $\Sigma^b_{i,k}$ and $\Pi^b_{i,k}$ are the smallest sets of formulæ satisfying the following inductive definition (where t is a term in language $\mathcal{L}_k$).

- $\Sigma^b_{0,k} = \Pi^b_{0,k}$ is the set of formulæ in language $\mathcal{L}_k$ in which all quantifiers are sharply bounded.
- $\Pi^b_{i,k} \subseteq \Sigma^b_{i+1,k}$.
- $\Sigma^b_{i,k} \subseteq \Pi^b_{i+1,k}$.
- If $A \in \Sigma^b_{i+1,k}$, then the formula $(\exists x \leq t)A$ is also in $\Sigma^b_{i+1,k}$.
- If $A \in \Pi^b_{i+1,k}$, then the formula $(\forall x \leq t)A$ is also in $\Pi^b_{i+1,k}$.
- If $A, B \in \Sigma^b_{i,k}$ (respectively $\Pi^b_{i,k}$), then so are $A \wedge B$ and $A \vee B$.
- If $A \in \Sigma^b_{i,k}$ (respectively $\Pi^b_{i,k}$), then $\neg A \in \Pi^b_{i,k}$ (respectively $\Sigma^b_{i,k}$).
- If $A \in \Sigma^b_{i,k}$ (respectively $\Pi^b_{i,k}$), then so are both $(\forall x \leq |t|)A$ and $(\exists x \leq |t|)A$.

When the language $\mathcal{L}_k$ is clear from context, particularly when $k = 2$, we omit the subscript k.

2.3 Theories of Bounded Arithmetic

The theory T^i_k is axiomatized by a fixed, finite set of quantifier-free axioms and the inference rule $\Sigma^b_{i,k} - IND$:

$$\frac{\Gamma(\mathbf{b}), A(a, \mathbf{b}) \longrightarrow A(a+1, \mathbf{b}), \Delta(\mathbf{b})}{\Gamma(\mathbf{b}), A(0, \mathbf{b}) \longrightarrow A(t(\mathbf{b}), \mathbf{b}), \Delta(\mathbf{b})} \quad ,$$

where $A \in \Sigma^b_{i,k}$ and $t \in \mathcal{L}_k$.

The theory S^i_k is axiomatized similarly, but with IND replaced by $\Sigma^b_{i,k} - PIND$ ("prefix induction"):

$$\frac{\Gamma(\mathbf{b}), A\left(\left\lfloor \frac{a}{2} \right\rfloor, \mathbf{b}\right) \longrightarrow A(a, \mathbf{b}), \Delta(\mathbf{b})}{\Gamma(\mathbf{b}), A(0, \mathbf{b}) \longrightarrow A(t(\mathbf{b}), \mathbf{b}), \Delta(\mathbf{b})} \quad ,$$

or an equivalent scheme named $LIND$, proven equivalent in [6, 1].

The theory R^i_k is axiomatized similarly, with one of four induction schemes, called DCI, $PPIND$, $LPIND$, and $LLIND$. They are shown to be equivalent in [1, 12] (in a sufficiently expressive language, such as $\mathcal{L}_k$ for $k \geq 2$). The functions Fh and Bh return the "front half" and "back half" respectively of a bit string.

$\Sigma^b_{i,k} - DCI$ ("divide and conquer induction"):

$$\frac{\Gamma(\mathbf{b}), A(Bh(a), \mathbf{b}), A(Fh(a), \mathbf{b}) \longrightarrow A(a, \mathbf{b}), \Delta(\mathbf{b})}{\Gamma(\mathbf{b}), A(0, \mathbf{b}), A(1, \mathbf{b}), A(2, \mathbf{b}), A(3, \mathbf{b}) \longrightarrow A(t(\mathbf{b}), \mathbf{b}), \Delta(\mathbf{b})} \quad .$$

The reader seeking more detail and discussion of these theories and induction schemes is referred to [6, 1, 12].

2.4 Circuits and Uniformity

Our complexity classes are defined in terms of uniform families of multiple-output circuits, which furthermore may contain "oracle gates". Wilson [25, 26] introduced a similar notion, and I follow his convention that an oracle gate's "size" and "depth" are its fanin and the log of its fanin respectively.

Most the circuit families in this paper are described by *genus*:

Definition 5. An oracle circuit family of *genus* k is a circuit family with at most $|\mathcal{L}_k|$ output gates, depth at most $||\mathcal{L}_k||$, and fanin at most 2 except for its oracle gates. All oracle gates in a given circuit family with input $\mathbf{x}$ have fanin exactly $|t(\mathbf{x})|$, where t is a nontrivial term in $\mathcal{L}_k$.

(By "nontrivial" we mean "not much smaller than any other term in $\mathcal{L}_k$": bounding t below by the maximum of its free variables suffices.)

Most results in this paper are stated "for $k \geq 2$" or "for $k \geq 3$". (Wilson's convention on the size and depth of oracle circuits makes no difference for $k \geq 3$.) Some would hold even for $k = 1$, but genus 1 circuit families are so ill-behaved (e.g., not closed under composition) that we shall ignore the $k = 1$ case.

Fact 6. *For $k \geq 2$, an oracle circuit family of genus k has size (i.e., number of gates) at most $|\mathcal{L}_k|$.*

All circuit families in this paper are assumed to be uniform in the U_{E^*} sense of Ruzzo [23]. However, we slightly extend Ruzzo's definition of the extended connection language to handle multiple-output circuits and oracle gates. The details appear in the journal version of this paper.

2.5 Complexity Classes

The polynomial-time hierarchy contains relation classes Σ_i^p, Π_i^p, and Δ_i^p, and function classes $\Box_i^p$ (see, e.g. [6]). Buss mentions in passing that analogous definitions and theorems hold if the language is expanded to $\mathcal{L}_k$, for $k \geq 2$. We therefore generalize the notation:

Definition 7. The complexity classes $\Sigma_{i,k}^p$, $\Pi_{i,k}^p$, $\Delta_{i,k}^p$, and $\Box_{i,k}^p$, for $k \geq 1$, are defined as follows:

- $\Sigma_{0,k}^p = \Pi_{0,k}^p = \Delta_{0,k}^p = \mathcal{P}$.
- For $i \geq 0$, $\Sigma_{i+1,k}^p$ is the closure of $\Pi_{i,k}^p$ under $\mathcal{L}_k$-bounded existential quantification.
- For $i \geq 0$, $\Pi_{i+1,k}^p$ is the closure of $\Sigma_{i,k}^p$ under $\mathcal{L}_k$-bounded universal quantification.
- For $i \geq 0$, $\Delta_{i+1,k}^p = \mathcal{P}^{\Sigma_{i,k}^p} = \mathcal{P}^{\Pi_{i,k}^p}$.
- $\Box_{0,k}^p = \mathcal{FP}$.
- For $i \geq 0$, $\Box_{i+1,k}^p = \mathcal{FP}^{\Sigma_{i,k}^p} = \mathcal{FP}^{\Pi_{i,k}^p}$.

For example, $\Delta^p_{1,2}$ is $\mathcal{P}$, $\Sigma^p_{1,2}$ is $\mathcal{NP}$, and $\Box^p_{2,3}$ is the class of functions computable in quasipolynomial time with oracles for nondeterministic quasipolynomial time. We also define analogous classes based on *parallel* complexity.

Definition 8. The complexity classes $\Sigma^c_{i,k}$, $\Pi^c_{i,k}$, $\Delta^c_{i,k}$, and $\Box^c_{i,k}$, for $k \geq 1$, are defined as follows:

- $\Sigma^c_{0,k} = \Pi^c_{0,k} = \Delta^c_{0,k} =$ the class of relations decidable by a uniform family of single-output circuits of genus k, containing no oracle gates.
- For $i \geq 0$, $\Pi^c_{i+1,k}$ is the closure of $\Sigma^c_{i,k}$ under $\mathcal{L}_k$-bounded universal quantification.
- For $i \geq 0$, $\Delta^c_{i+1,k}$ is the class of relations decidable by a uniform family of single-output circuits of genus k, containing oracle gates for some $\Sigma^c_{i,k}$ or $\Pi^c_{i,k}$ relation.
- $\Box^c_{0,k}$ is the class of functions computable by a uniform family of $|\mathcal{L}_k|$-output circuits of genus k, with no oracle gates.
- For $i \geq 0$, $\Box^c_{i+1,k}$ is the class of functions computable by a uniform family of $|\mathcal{L}_k|$-output circuits of genus k, containing oracle gates for some $\Sigma^c_{i,k}$ or $\Pi^c_{i,k}$ relation.

Henceforth families with the uniformity, size, depth, and gate restrictions above will be called $\Sigma^c_{i,k}$, $\Pi^c_{i,k}$, and $\Delta^c_{i,k}$ circuit families respectively.

The main theorem of Sect. 5 doesn't appear to hold in its most elegant form for $k = 2$ — indeed, if it did, it would imply $\mathcal{NC} = \mathcal{NC}^1$. However, we can prove a less elegant analogue for complexity classes defined in terms of $\mathcal{NC}$, with the depth bounds of $\Box^c_{0,3}$ and the size and fanin bounds of $\Box^c_{0,2}$.

Definition 9 (Allen). $\mathcal{FNC}$ comprises the functions computable by U_{E^*}-uniform[2] circuit families with polylog depth, polynomial size, and polynomially many outputs.

Definition 10. For any class X of relations on $\{0,1\}^*$,

- $\mathcal{FNC}^X$ is the class of functions computable by U_{E^*}-uniform, polylog depth, polynomial size, oracle circuit families with oracle gates of fanin $|t|$ (where $t \in \mathcal{L}_2$) for an X relation.
- $\mathcal{NC}^X$ is the class of relations whose characteristic functions are 0/1-valued $\mathcal{FNC}^X$ functions.

As usual, an oracle gate's "depth" and "size" are considered to be $||t||$ and $|t|$ respectively.

Normally we shall study classes of the form $\mathcal{NC}^{\Sigma^c_{i-1,2}}$ and $\mathcal{FNC}^{\Sigma^c_{i-1,2}}$, by analogy to $\Delta^c_{i,k}$ and $\Box^c_{i,k}$.

[2] Allen actually defined $\mathcal{FNC}$ circuit families to be logspace uniform, but as he points out, it makes no difference at this level;see [23].

2.6 Alternating Turing Machines

We can also define an alternating Turing machine model to correspond to arbitrary levels in the Δ^c hierarchies.

Definition 11. An *oracle alternating Turing machine of genus k* is an alternating $||\mathcal{L}_k||$-time Turing machine whose states are partitioned into universal, existential, exclusive-or, and oracle states. The universal, existential, and exclusive-or states each have exactly two next states. The behaviour of all oracle states in a given machine depends on a global "oracle bound" $t \in \mathcal{L}_k$, "nontrivial" in the same sense as defined in Sect. 2.4.

When the machine enters an oracle state for relation ρ, it guesses a binary number $0 \leq j < |t(x)|$, writes it and a delimiter at the end of a special oracle tape, and goes on to a next state deterministically. Let D_j denote the accept/reject decision of this next state given the number j; then the oracle state accepts iff the relation ρ holds of the $|t|$-bit string $D_{|t|-1}D_{|t|-2}\cdots D_2D_1D_0$. An oracle state is counted as taking $||t||$ time steps.

(Recall that Chandra, Kozen, & Stockmeyer [10] defined ATM's with NOT states, then proved that they could be eliminated. This proof appears not to work in the presence of oracle states, unless the oracles come from a class closed under complement. We use two-input exclusive-or states instead, as they can be "programmed" to compute either a NOT or an identity function by making one of their successor configurations unconditionally accept or reject.)

For example, if ρ were a $|t|$-way AND, a ρ-oracle state would be equivalent to universally guessing $||t||$ bits, which could have equally well been done without an oracle in the same time. Similar reasoning holds for any $\rho \in \Delta^c_{0,k}$; thus we have the

Lemma 12. *For $k \geq 2$, a relation is computable by an oracle ATM of genus k with a $\Delta^c_{0,k}$ oracle iff it is computable by an ATM of genus k with no oracles at all.*

A more interesting situation arises when ρ is, say, a $\Sigma^c_{1,k}$ relation; then there is no obvious way to simulate the oracle in less than $|\mathcal{L}_k|$ parallel time. The following lemma, whose proof appears in the journal version, may be seen as a generalization of Ruzzo's theorems 3 and 4.

Lemma 13. *For $k \geq 2$, a relation is computable by an ATM of genus k with an oracle in $\Sigma^c_{i-1,k}$ iff the relation is in $\Delta^c_{i,k}$.*

In particular, $\Delta^c_{0,k}$ is equal to $ATIME(||\mathcal{L}_k||)$ for all $k \geq 2$. For example, $\Delta^c_{0,2}$ is known in the literature as ALOGTIME or uniform $\mathcal{NC}^1$, and $\Delta^c_{0,3}$ is the class of problems solvable in uniform polylog depth, or alternating polylog time, or deterministic polylog space (for other relevant characterizations of these classes, see [4]).

3 Basic Relationships among the Classes

In this section we state "the easy theorems" about the classes defined in the previous section, showing they constitute a well-behaved complexity hierarchy, before proceeding to more profound results. Again, the proofs (all quite straightforward) appear in the journal version of this paper.

Lemma 14. *For $k \geq 2$, $\Delta^c_{0,k} \subseteq \Delta^p_{1,k}$.*

Theorem 15. *For $i \geq 1, k \geq 2$, $\Sigma^c_{i,k} = \Sigma^p_{i,k}$ and $\Pi^c_{i,k} = \Pi^p_{i,k}$.*

This follows immediately from $\Sigma^c_{1,k} = \Sigma^p_{1,k}$, which in other notation has been known at least since 1974 [14]. In light of this theorem, we can dispense with proving basic properties of the $\Sigma^c_{i,k}$ and $\Pi^c_{i,k}$ classes: either we already know them, or we are unlikely to be able to prove them with present techniques. Indeed, we can dispense with the notations $\Sigma^c_{i,k}$ and $\Pi^c_{i,k}$ altogether. But we still don't know much about the $\Delta^c_{i,k}$ and $\Box^c_{i,k}$ classes. So we'll state a number of useful results about these classes and the relationships among them.

Lemma 16.

- *The $\Delta^c_{i,k}$ relations are the relations with characteristic functions in $\Box^c_{i,k}$.*
- *For $i \geq 0$, $k \geq 1$, $\Delta^c_{i,k}$ is closed under negation, finite disjunction and conjunction, and definition by $\Delta^c_{i,k}$ cases (and similarly with $\mathcal{NC}^X$ in place of $\Delta^c_{i,k}$).*
- *For all $i' \geq i \geq 0$ and $k' \geq k \geq 1$, we have $\Delta^c_{i,k} \subseteq \Delta^c_{i',k'}$ and $\Box^c_{i,k} \subseteq \Box^c_{i',k'}$.*
- *$\Sigma^c_{i,k} \cup \Pi^c_{i,k} \subseteq \Delta^c_{i+1,k}$.*

Lemma 17. *For $i \geq 0$ and $k \geq 2$, $\Box^c_{i,k}$ and $\mathcal{FNC}^X$ are closed under composition.*

Corollary 18. *For $i \geq 0$ and $k \geq 2$, $\Delta^c_{i,k}$ is closed under substitution by $\Box^c_{i,k}$ functions (and similarly with $\mathcal{NC}^X$ and $\mathcal{FNC}^X$ replacing $\Delta^c_{i,k}$ and $\Box^c_{i,k}$ respectively).*

Lemma 19. *For $k \geq 2$, $\Delta^c_{0,k} = \Delta^c_{1,k}$.* *(Similarly, $\mathcal{NC} = \mathcal{NC}^{\mathcal{NC}}$.)*

The proof of this lemma for $k = 2$ depends on the convention that an oracle gate with fanin $|t|$ is counted as having "depth" $||t||$. The proof for $\mathcal{NC}$ depends on the convention that an oracle gate is counted as having "size" $|t|$.

Lemma 20. *For any $i > 0$, $k \geq 2$, the function $f(\mathbf{x})$ is in $\Box^c_{i,k}$ iff its* bit-graph, *the relation $\lambda j, \mathbf{x}.Bit(j, f(\mathbf{x}))$, is in $\Delta^c_{i,k}$.*

Similarly, a function is in $\mathcal{FNC}^X$ iff its bit-graph is in $\mathcal{NC}^X$.

Lemma 21. *For $i \geq 0$ and $k \geq 2$, if $\rho(\mathbf{x})$ is equivalent to $(\exists y \leq |s(\mathbf{x})|)\sigma(\mathbf{x}, y)$ or to $(\forall y \leq |s(\mathbf{x})|)\sigma(\mathbf{x}, y)$, where $\sigma \in \Delta^c_{i,k}$ and $s \in \mathcal{L}_k$, (respectively $\sigma \in \mathcal{NC}^X$, with $s \in \mathcal{L}_2$), then ρ is itself in $\Delta^c_{i,k}$ (respectively $\mathcal{NC}^X$).*

That is, $\Delta^c_{i,k}$ and $\mathcal{NC}^X$ are is closed under sharply-bounded quantifiers.

Lemma 22. *For $i \geq 0, k \geq 2$, $\Delta^c_{i,k} \subseteq \Delta^p_{i,k}$. (Similarly, $\mathcal{NC}^{\Sigma^p_{i-1,2}} \subseteq \Delta^p_{i,2}$.)*

Corollary 23. *For $i \geq 0, k \geq 2$, $\Delta^c_{i,k} \subseteq \Sigma^p_{i,k} \cap \Pi^p_{i,k}$.
(Similarly, $\mathcal{NC}^{\Sigma^p_{i-1,2}} \subseteq \Sigma^p_{i,2} \cap \Pi^p_{i,2}$.)*

Corollary 24. *For $i \geq 0$, $k \geq 2$, $\Sigma^p_{i,k}$ and $\Pi^p_{i,k}$ are closed under definition by $\Delta^c_{i,k}$ and $\mathcal{NC}^{\Sigma^p_{i-1,2}}$ cases.*

Lemma 25. *For $i \geq 1, k \geq 2$, $\square^c_{i,k} \subseteq \square^p_{i,k}$ and $\mathcal{FNC}^{\Sigma^p_{i-1,2}} \subseteq \square^p_{i,2}$.*

Corollary 26. *For $i \geq 0$, $k \geq 2$, $\Sigma^p_{i,k}$ and $\Pi^p_{i,k}$ are closed under substitution by $\square^c_{i,k}$ (and $\mathcal{FNC}^{\Sigma^p_{i-1,2}}$) functions.*

Theorem 27. *For $i \geq 1, k \geq 2$ we have the inclusions $\Delta^c_{i,k} \subseteq \Delta^p_{i,k} \subseteq \Delta^c_{i+1,k}$ and $\square^c_{i,k} \subseteq \square^p_{i,k} \subseteq \square^c_{i+1,k}$.*

4 The Buss-Krajíček-Takeuti Characterization

Buss, Krajíček, and Takeuti [9] give a characterization of the $\Sigma^b_{i,3}$-definable functions of R^i_3. The computational model in that paper uses "witnessing oracles", which answer an existential query either with "no" or by providing a satisfying value (of quasipolynomial length) for the outermost existential quantifier. Since there may be multiple possible witnesses to a given existential question, the functions defined by invoking these witnessing oracles may be multi-valued. Nevertheless, in certain cases this computational model is equivalent to the parallel model of the present paper.

The main result of this section, Theorem 31, is a direct proof that the function class $\mathcal{FP}_3^{\Sigma^p_{i-1}}[wit, \log^{O(1)}]$ of Buss, Krajíček, and Takeuti, restricted to single-valued functions, is precisely $\square^c_{i,3}$. The same result could also be proven from Theorem 35 together with results of [9].

Definition 28 (Buss, Krajíček, Takeuti). A multi-valued function f is computable in $\mathcal{FP}_3^{\Sigma^p_{i-1}}[wit, \log^{O(1)}]$ iff there is a Turing machine that runs in quasipolynomial time and makes at most polylog many queries to a fixed Σ^p_{i-1} witnessing oracle, such that for all x, M outputs one of the values of $f(x)$.

The notation $\mathcal{FP}_3^{\Sigma^p_{i-1}}$, without the stuff in brackets, is what I call $\square^p_{i,3}$. Note that the oracle may (in my notation) be $\Sigma^p_{i-1,k}$ for *any* $k \leq 3$, without changing the class thus defined: the machine making the query can precompute the quantifier bounds and pass them to the oracle as extra parameters.

Definition 29 (Buss, Krajíček, Takeuti). A multi-valued function f is Σ^b_i-defined by theory T iff for some Σ^b_i formula A,

- $T \vdash (\forall x)(\exists y)A(x, y)$, and
- if $A(n, m)$ holds in the natural numbers, then m is a value of $f(n)$.

(Note that the second requirement does *not* say "if and only if". Buss, Krajíček, and Takeuti also defined a class called "strong $\mathcal{FP}_3^{\Sigma^p_{i-1}}[wit, \log^{O(1)}]$" and a notion of "strong Σ^b_i-definability" using "if and only if", but the difference is only relevant for multi-valued functions and so we shall ignore it henceforth.)

Theorem 30 (Buss, Krajíček, Takeuti). *For $i \geq 2$, a multi-valued function f is in $\mathcal{FP}_3^{\Sigma^p_{i-1,3}}[wit, \log^{O(1)}]$ iff it is $\Sigma^b_{i,3}$-defined by theory R^i_3.*

Theorem 31. *For $i \geq 2$ and $k \geq 3$, the restriction of $\mathcal{FP}_k^{\Sigma^p_{i-1,k}}[wit, ||\mathcal{L}_k||]$ to single-valued functions is $\Box^c_{i,k}$.*

Proof. ($\supseteq$): Suppose $f \in \Box^c_{i,k}$. Then f is computable by a uniform $\Box^c_{i,k}$ circuit family with $\Sigma^p_{i-1,k}$ yes/no oracle ρ.

With at most a constant factor increase in depth, we can assume the circuit family is levelled, that all output gates are at level 0, that each even level contains no oracle gates, and that each odd level consists entirely of oracle gates. (The last restriction requires excluding constant-valued ρ.) Embed the resulting gates in a rectangular array, indexed by row and column (r, c), in such a manner that a gate's position in the array easily determines its gate number and thence its type. Thus the resulting circuit family is still U_{E^*} uniform.

The output of gate (r, c) can be described by a $\Sigma^p_{i-1,k}$ relation $\sigma(n, r, c, \mathbf{b})$ where n is the input size and $\mathbf{b}$ are the outputs of row $r + 1$. (The relation σ simply determines the type of gate (r, c) and simulates it; the most complex case is that of an oracle gate, which is $\Sigma^p_{i-1,k}$.) Define $\phi(n, r, \mathbf{b}, j)$ to be the formula

$$(\exists |w| = \text{width of row } r)(|w|_1 \geq j) \wedge (\forall c < |w|)(Bit(c, w) \supset \sigma(|x|, r, c, \mathbf{b}))) \quad ,$$

where $|w|_1$ represents the number of 1's in w. The ϕ relation is $\Sigma^b_{i-1,k}$ (note $i \geq 2$), and monotone in j, so for any fixed x, we can find the maximum j satisfying ϕ by binary search in $||\mathcal{L}_k||$ time. For this maximum j, the only possible witness w is the string representing *exactly* the values of all the gates at level r (assuming $\mathbf{b}$ correctly represents the values of all the gates at level $r+1$), because the clause $(\forall c < |w|)(Bit(c, w) \supset \sigma(|x|, r, c, \mathbf{b}))$ ensures that all the gates whose bit in w is 1 are correctly computed, and if any gate whose bit in w is 0 were incorrectly computed, j wouldn't be maximal.

Thus $||\mathcal{L}_k||$ ϕ-queries, the last one witnessing, suffice to determine the outputs of all gates at level r from the outputs of all gates at level $r + 1$. Iterating this over the $||\mathcal{L}_k||$ levels, starting with the input, allows us to compute the correct values of all the gates at the output level, and thus the value of the function.

($\subseteq$): Suppose $f \in \mathcal{FP}_k^{\Sigma^p_{i-1,k}}[wit, ||\mathcal{L}_k||]$ and f is single-valued. As shown in [9], we can assume without loss of generality that the quasipolynomial-time machine only uses an ordinary yes/no oracle until its last query, at which point it demands a witness. Furthermore, we can assume without loss of generality that it doesn't demand this witness until it has confirmed by yes/no query that there is one.

Let formula $\psi(x, w)$ assert that there is a computation of the $\mathcal{FP}_k$ machine (up to the time that it demands a witness) in which

- the computation starts with x on the input,
- for all j, if the j-th most significant bit of w is 1, then the j-th oracle query in the computation answers "yes",
- the behaviour of the machine is consistent with the alleged tape contents and alleged oracle answers in the computation, and
- all the "yes" oracle answers (at most $||\mathcal{L}_k||$) in the computation are correct.

The formula $\psi(x, w)$ is $\Sigma^b_{i-1,k}$ (note $i \geq 2$), consisting of a bounded existential quantifier around some sharply-bounded universal quantifiers around a $\Sigma^p_{i-1,k}$ oracle. Furthermore, ψ is monotone in each bit of w. Since w has only $||\mathcal{L}_k||$ many bits, we can find the lexicographically maximum w satisfying ψ by deterministic binary search. Having found this w, for each output bit r in parallel we query the $\Sigma^p_{i-1,k}$ oracle $\psi'(x, w, r)$, which asserts that there is a computation in which

- the computation starts with x on the input,
- for all j, the j-th most significant bit of w is exactly the answer to the j-th oracle query in the computation,
- the behaviour of the machine is consistent with its alleged tape contents and the alleged oracle answers,
- all the "yes" oracle answers in the computation are correct, and
- there exists a witness to the final query that leads the machine to output a value with bit r set.

By maximality of w, there is a unique and correct such computation until the final witnessing query. Since we assumed f was single-valued, all the output bits will be consistent, even if the circuits to compute different bits happen to find different witnesses to the final oracle query. Thus $f \in \Box^c_{i,k}$. □

Krajíček [16] shows that the predicates in $\mathcal{P}^{\Sigma^p_{i-1}}[O(\log)]$ (i.e., $\mathcal{P}$ with $O(\log)$ queries to a Σ^p_{i-1} oracle) are precisely those Σ^b_i-definable in S^{i-1}_2. His technique actually uses $\mathcal{FP}^{\Sigma^p_{i-1}}[wit, O(\log)]$, but the restriction to predicates rather than functions allows him to eliminate the witnessing and multivaluedness at the last moment. Interestingly enough, the above proof adapts to $k = 2$:

Theorem 32. *For $i \geq 2$, the restriction of $\mathcal{FP}^{\Sigma^p_{i-1}}[wit, O(\log)]$ to single-valued functions is $\Box^c_{i,2}$.*

Proof. The proof of Theorem 31 relies on $k \geq 3$ in several ways, but these can all be fixed.

($\supseteq$): The previous proof takes $||\mathcal{L}_k||$ time to do a binary search for each of the $||\mathcal{L}_k||$ rows of the array. Since $||\mathcal{L}_2||$ is not closed under multiplication, we can no longer afford this. However, by our convention, for some term $t \in \mathcal{L}_2$, each oracle gate has "depth" $||t|| = \theta(\log(n))$. But a $\Box^c_{i,2}$ circuit has depth $O(\log(n))$, so there must be a constant bound d on the number of oracle gates along any one path. Arrange the array in d meta-layers, each containing one row of oracle gates and $O(\log)$ rows of ordinary gates. We define a $\Delta^c_{0,2}$ relation $\sigma'(n, r, c, \mathbf{b})$ that does the same thing as σ for rows containing only ordinary gates. We then define a $\Sigma^p_{1,2}$ relation $\phi'(n, r, \mathbf{b})$ by $(\exists |w| = \text{width of row } r)(\forall c < |w|)(Bit(c, w) \leftrightarrow \sigma'(|x|, r, c, \mathbf{b})))$. The algorithm is then as before, but for ordinary rows we simply make a witnessing query to ϕ' rather than doing a binary search on ϕ.

($\subseteq$): We can still assume an $\mathcal{FP}^{\Sigma^p_{i-1}}[wit, O(\log)]$ machine demands witnesses only on its final query, by a lemma in [16]. The rest of the proof is unaltered. □

Corollary 33. *For $i \geq 2$, $k \geq 2$, the functions $\Sigma^b_{i,k}$-definable in S^{i-1}_k are $\Box^c_{i,k}$.*

Corollary 34. *For $i \geq 2$, $k \geq 2$, if $S^{i-1}_k = T^{i-1}_k$ then $\Box^c_{i,k} = \Box^p_{i,k}$.*

5 Links to Bounded Arithmetic

We next state and prove a theorem analogous to that in [6], demonstrating a close link between computation in this parallel hierarchy and the bounded-arithmetic theories R^i_k. As a result, we can simplify and strengthen the conservation results of [9].

Theorem 35. *For $i \geq 1, k \geq 3$, $\Box^c_{i,k}$ contains exactly the $\Sigma^b_{i,k}$-definable functions of R^i_k.*
For $i \geq 1$, $\mathcal{FNC}^{\Sigma^p_{i-1,2}}$ contains exactly the $\Sigma^b_{i,2}$-definable functions of R^i_2.

This is proven by methods similar to those in [6, 1]. First, $\Box^c_{i,k}$ (respectively $\mathcal{FNC}^{\Sigma^p_{i-1,2}}$) is equal to a certain recursively-defined function algebra. Second, the base functions of this algebra are provably total in R^i_k, and the operators preserve provable totality, so any function in the algebra is provably total in R^i_k. Third, any $\Sigma^b_{i,k} \cup \Pi^b_{i,k}$ sequent provable in R^i_k can be witnessed by a function in this algebra.

5.1 A Function Algebra

We define a hierarchy of function algebras $\{A_{i,k}\}_{i \geq 0, k \geq 2}$ by applying two simple operations to a fixed collection of base functions on the universe of finite bit-strings.

Definition 36. The *BASE* functions are the following:

- 0, 1, λ, the constant functions (λ represents the empty string),
- $Lsp(x, y)$, the "least significant part" function, defined to be the rightmost $|y|$ bits of x, padded on the left with zeroes if $|y| > |x|$.
- $Msp(x, y)$, the "most significant part" function, defined to be all but the rightmost $|y|$ bits of x, or the empty string if $|y| > |x|$.

- $Cond(w,x,y,z) = \begin{cases} x & \text{if } w = \lambda \\ y & \text{if } Lsp(w,1) = 0 \\ z & \text{if } Lsp(w,1) = 1 \end{cases}$
- $Conc(x,y)$, the concatenation of x and y,
- $Bh(x)$, the "back half" function, defined to be the rightmost $\left\lceil \frac{|x|}{2} \right\rceil$ bits of x,
- $Fh(x)$, the "front half" function, defined to be the leftmost $\left\lfloor \frac{|x|}{2} \right\rfloor$ bits of x,
- $Not(x)$, the one's complement of x,
- $Or(x,y)$, the bitwise OR of x and y (undefined if $|x| \neq |y|$),
- $And(x,y)$, the bitwise AND of x and y (undefined if $|x| \neq |y|$),
- $Ins_0(x)$, a string exactly twice as long as x such that for all $i < |x|$, we have $Bit(2i, Ins_0(x)) = 0$ and $Bit(2i+1, Ins_0(x)) = Bit(i,x)$,
- $Ins_1(x)$, defined analogously with $Bit(2i, Ins_1(x)) = 1$.

Definition 37. A function f is defined by *$|\mathcal{L}_k|$-bounded divide-and-conquer recursion* from functions g and h if there is a term $t \in \mathcal{L}_k$ such that

$$f(z,b,\mathbf{x}) = \begin{cases} g(z,\mathbf{x}) & \text{if } |z| \leq |b|, \\ h(z,b,\mathbf{x}, f(Fh(z),b,\mathbf{x}), f(Bh(z),b,\mathbf{x})) & \text{otherwise,} \end{cases}$$

and $|f(z,b,\mathbf{x})| \leq |t(z,b,\mathbf{x})|$ for all $z, b, \mathbf{x}$.

(The somewhat inelegant $|t|$ bound can be eliminated by the use of "tiered recursion", as in [2, 4, 19]. For the purposes of this paper, the arbitrary bound is less onerous than the additional notation and explanation needed for tiered descriptions of these function classes.)

Definition 38. The function algebra $A_{i,k}$ is the closure of $BASE$ and the characteristic functions of $\Sigma^p_{i-1,k}$ relations[3] under the operations of composition and $|\mathcal{L}_k|$-bounded divide-and-conquer recursion.

For example, we define a function $CountUp(s) \in A_{0,2}$ which rounds $|s|$ up to the next power of 2, and produces the concatenation of the binary numbers $0, 1, \ldots, |s| - 1$, each embedded in a block of $||s||$ bits. (This example not only shows some of DCR's power, but proves useful later on.) Let $SHL(x,y) = Lsp(Conc(x,y),x)$; this appends y to x, discarding high bits so the result still has length $|x|$. Let $Zeroes(x) = Lsp(0,x)$, a string of zeroes of length $|x|$. Let

$$Msk(z,b) = \begin{cases} \lambda & \text{if } |z| = 0 \\ SHL(Zeroes(b),1) & \text{if } |z| = 1 \\ Conc(SHL(Msk(Bh(z),b),0), SHL(Msk(Bh(z),b),0)) & \text{if } |z| > 1, \end{cases}$$

$$Aux(z,b) = \begin{cases} \lambda & \text{if } |z| = 0 \\ Zeroes(b) & \text{if } |z| = 1 \\ Conc(Aux(Bh(z),b), Or(Msk(Bh(z),b), Aux(Bh(z),b))) & \text{if } |z| > 1, \end{cases}$$

and finally $CountUp(z) = Aux(z,|z|)$. I leave seeing how this works as an exercise for the reader.

[3] where $\Sigma^p_{-1,k}$ is by convention {}

Lemma 39. *For $i \geq 0$ and $k \geq 3$, $\Box^c_{i,k} = A_{i,k}$.*
For $i \geq 0$ and $k = 2$, $\Box^c_{i,k} \subseteq A_{i,k} \subseteq \mathcal{FNC}^{\Sigma^p_{i-1,k}}$.

Proof. First we consider the $i = 0$ case.

($\supseteq$): Suppose $f \in A_{0,k}$. All the *BASE* functions are clearly in $\Box^c_{0,k}$. By Lemma 17, $\Box^c_{0,k}$ is closed under composition, and it is not hard to see that it's closed under $|\mathcal{L}_k|$-bounded divide-and-conquer recursion too. (This requires $k \geq 3$ so circuit depth is closed under multiplication. For the $k = 2$ case, since $\mathcal{FNC}$ is closed under polynomial-bounded DCR [1], we can conclude $A_{0,2} \subseteq \mathcal{FNC}$.)

($\subseteq$): Suppose $f \in \Box^c_{0,k}$ and consider its $\Delta^c_{0,k}$ bit-graph. Recall that $\Delta^c_{0,k} = ATIME(||\mathcal{L}_k||)$. At the cost of a constant factor in time, we can assume the ATM is *strictly* alternating between existential and universal states and that it accesses its input tape only at leaves. In [4], the author showed how to simulate the computation of such an ATM within a function algebra similar to $A_{0,k}$. In brief: construct a binary tree data structure called PATHS, each of whose leaves contains an encoding of the path from the root to that leaf; apply $\mathcal{L}_k$-bounded divide-and-conquer recursion to this encoded path to determine the ATM configuration at the corresponding leaf of the computation tree; and combine the Boolean results using a constant-bounded divide-and-conquer recursion. Thus $\Delta^c_{0,k} \subseteq A_{0,k}$. Now if the function value is $|s|$ bits long, it can be computed from the bit-graph by $|s|$-bounded divide-and-conquer recursion on $CountUp(s)$, computing the i-th bit at leaf i and concatenating the results together, again as in [4]. Thus $\Box^c_{0,k} \subseteq A_{0,k}$.

For $i > 0$, the $\supseteq$ direction, recall that the characteristic functions of $\Sigma^p_{i-1,k}$ relations are in $\Box^c_{i,k}$ (and $\mathcal{FNC}^{\Sigma^p_{i-1,k}}$) by definition. The proof that $A_{i,k} \subseteq \Box^c_{i,k}$ (for $k = 2$, $\mathcal{FNC}^{\Sigma^p_{i-1,2}}$) is almost exactly as before.

The $\subseteq$ direction is a little more complex. Recall from Lemmas 13 and 20 that every $\Box^c_{i,k}$ function's bit-graph is in $ATIME(||\mathcal{L}_k||)$ with oracle states for $\Sigma^p_{i-1,k}$ relations. At the cost of a constant factor in time, we assume the levels of the ATM rotate strictly among universal, existential, parity, and oracle states. Let t denote the bounding term for all the oracle states in the machine, rounded up so that $|t|$ is a power of 2. We construct a PATHS tree of depth equal to the time bound of the ATM (counting oracles as taking $||t||$ time, as usual).

Finally, we define an $A_{i,k}$ function $EVALTREE(T, b, x)$ by divide and conquer recursion. (We shall eventually invoke it with T equal to the PATHS tree, b any string the length of one leaf of the PATHS tree, and x equal to the original input.) As defined in [4], *EVALTREE* returned a bit indicating whether a certain subtree of the ATM's computation tree accepted or rejected. In this setting, it will ultimately do the same, but it may have intermediate values up to $|t(x)|$ bits long. The base case of the recursion, exactly as in [4], treats T as encoding the path to a particular leaf of the ATM's computation tree; it follows this path, applying the left or right next-step function at each step, to construct the configuration of that leaf, and finally decides whether that leaf accepts or rejects.

Since $|T|$ is cut in half at each iteration, $||T||$ indicates the depth of a tree, up to an additive constant. Of every $||t|| + 3$ levels of the PATHS tree, $||t||$ are used to accumulate the answers from the $|t|$ children of a given oracle configuration, one computes parity, one OR, and one AND. The final value of *EVALTREE* is 1 or 0, indicating whether the ATM accepts or rejects. The value of a multi-bit function can be pieced together just as in the $i = 0$ case.

□

This concludes the proof of lemma 39.

Lemma 40. *For $i \geq 1$, $A_{i,2} = \mathcal{FNC}^{\Sigma^p_{i-1,2}}$.*

Proof. Lemma 39 has already given us the $\subseteq$ direction. For the $\supseteq$ direction, which resembles the proof of Allen's Theorem 1.3.3 [1], let $f \in \mathcal{FNC}^{\Sigma^p_{i-1,2}}$ be computed by a (wolog, levelled) circuit family of depth $O(\log^j(n))$. We reason by induction on j.

If $j = 1$, then $f \in \Box^c_{i,2}$, and therefore $f \in A_{i,2}$ by Lemma 39. If $j > 1$, think of an f circuit as divided into $\log(n)$ meta-layers of depth at most $\log^{j-1}(n)$ each. Let g be a function which, given the values of all the inputs to a meta-layer and a number indicating which meta-layer it is, computes all the outputs from that meta-layer. This function can be computed in depth $O(\log^{j-1}(n))$, even allowing for constructing a copy of the relevant part of the f circuit, so by the inductive hypothesis $g \in A_{i,2}$. The desired function f can then be computed by iterating g $O(\log(n))$ times, which we can do by DCR. Since the f circuit is only polynomial size, each value of f and g need only be polynomially many bits long, so the DCR is polynomially bounded. □

5.2 Definability in R^i_k

Theorem 41. *For $i \geq 1$, $k \geq 2$, every $A_{i,k}$ function is $\Sigma^b_{i,k}$-definable in R^i_k.*

All the *BASE* functions, and the characteristic functions of $\Sigma^p_{i-1,k}$ relations, are provably total in R^i_k for $i \geq 1, k \geq 2$. The class of functions $\Sigma^b_{i,k}$-definable in R^i_k is trivially closed under composition, so we need only to prove it closed under $|\mathcal{L}_k|$-bounded divide-and-conquer recursion.

Lemma 42. *For $i \geq 1$, $k \geq 2$, if g and h are $\Sigma^b_{i,k}$-definable in R^i_k, and f is defined by $|\mathcal{L}_k|$-bounded divide-and-conquer recursion on them, then f is $\Sigma^b_{i,k}$-definable in R^i_k.*

Proof. Suppose g and h are defined in R^i_k by the Σ^b_i formulæ $\phi_g(z, \mathbf{x}, y)$ and $\phi_h(z, b, \mathbf{x}, u_1, u_2, y)$ respectively, and f is defined from them with a length bound of $|s|$. We define a formula $\phi_f(z, b, \mathbf{x}, y)$ that $\Sigma^b_{i,k}$-defines f in R^i_k by asserting the existence of an encoded binary tree T representing the computation. At vertex i of this binary tree we'll store data of the form $\langle y_i, z_i \rangle$ (of length $O(|s| + |z|)$) to assert that $f(z_i, b, \mathbf{x}) = y_i$. The root block must be equal to $\langle y, z \rangle$, each block $\langle y_i, z_i \rangle$ with $|z_i| > |b|$ must be appropriately related to its children by ϕ_h, and each block $\langle y_i, z_i \rangle$ with $|z| \leq |b|$ must satisfy $\phi_g(z_i, \mathbf{x}, y_i)$.

In order to ensure that R^i_k can prove the existence of such a T, we represent T in a somewhat inefficient, "inflated" way: to represent a tree of depth $||z||$ with $O(|s|+|z|)$ bits of information at each vertex, we use an array of $4^{||z||-1}$ blocks, each of length $O(|s|+|z|)$. Each subtree is represented by a string whose first half contains the data for the subtree's root, padded with garbage bits of unspecified value. The third quarter of the string represents the left subtree, and the fourth quarter the right subtree. This enables us to apply divide-and-conquer induction easily to the whole tree; for details of the techniques, see [4]. Note that even in inflated form, the length of T is a polynomial in $|z|$ and $|s|$, so this construction works for $k \geq 2$. □

Theorem 41 follows immediately.

5.3 Witnessing R^i_k Proofs

It remains only to show that every function $\Sigma^b_{i,k}$-definable in R^i_k is in $A_{i,k}$. This is done by a witnessing theorem similar to those of [6, 1]. I assume familiarity with their techniques, particularly the *Witness* predicate. To extend those techniques to the present setting, we need

Lemma 43. *For $i \geq 1, k \geq 2$, if $A(\mathbf{a}) \in \Sigma^b_{i,k}$, then $\mathit{Witness}^{i,\mathbf{a}}_A$ is in $\Delta^c_{i,k}$.*

Buss proved that it was a $\Delta^p_{i,k}$ predicate, but we have no evidence that $\Delta^p_{i,k} = \Delta^c_{i,k}$. Allen stated and proved the lemma for $i = 1, k = 2$, but there is no essential reason for this restriction. The proof is a straightforward induction on the syntax of A, relying on $\Box^c_{i,k}$'s ability to do simple sequence coding and decoding. It uses nothing more sophisticated than Lemmas 18 and 21.

Theorem 44. *For $i \geq 1$, $k \geq 2$, if $R^i_k \vdash \Gamma(\mathbf{a}) \longrightarrow \Delta(\mathbf{a})$, where Γ and Δ are lists of (0 or more) $\Sigma^b_{i,k}$ formulæ with free variables among $\mathbf{a}$, then there is an $A_{i,k}$ function f such that*

$$R^i_k \vdash \mathit{Witness}^{i,\mathbf{a}}_{\bigwedge \Gamma}(w,\mathbf{a}) \supset \mathit{Witness}^{i,\mathbf{a}}_{\bigvee \Delta}(f(w,\mathbf{a}),\mathbf{a}) \quad .$$

Proof. All the essential ideas appear in either [6] or [1]. Buss shows how to witness axioms and formulæ derived by cosmetic, Boolean, and quantifier inferences. All of these can still be handled within $\Box^c_{0,k}$, which we know is contained in $A_{0,k}$ for $k \geq 2$. Allen shows how to use divide-and-conquer recursion to provably witness a formula in whose proof the last step is Σ^b_1-DCI; the proof extends to Σ^b_i-DCI by Lemma 43. □

This concludes the proof of Theorem 35.

Since S^i_k $\Sigma^b_{i,k}$-defines precisely the $\Box^p_{i,k}$ functions, we immediately get

Corollary 45. *For $i \geq 1$, $k \geq 3$, if S^i_k is $\forall\exists\Sigma^b_{i,k}$-conservative over R^i_k, then $\Box^c_{i,k} = \Box^p_{i,k}$.*

Corollary 46. *For $i \geq 1$, $k = 2$,if S^i_k is $\forall\exists\Sigma^b_{i,k}$-conservative over R^i_k, then $\mathcal{FNC}^{\Sigma^p_{i-1,k}} = \Box^p_{i,k}$.*

5.4 Conservation, Pro and Con

The characterization of Theorem 35 leads to a new proof of Theorem 21 of [9]:

Theorem 47 (Buss, Krajíček, Takeuti). *For $i \geq 2$, $k \geq 3$, R^i_k is conservative over S^{i-1}_k with respect to $\forall\exists\Sigma^b_{i,k}$ sentences.*

Since the $\forall\exists\Sigma^b_{i,k}$ consequences of R^i_k correspond exactly to the $\Box^c_{i,k}$ functions, it suffices to show that S^{i-1}_k can $\Sigma^b_{i,k}$-define the $\Box^c_{i,k}$ functions too. We stated this before, in Corollary 33, but that depended on the result we're now trying to prove. Here's a direct proof (which incidentally works for $k = 2$ as well).

Lemma 48. *For $i \geq 2$, $k \geq 2$, S^{i-1}_k can $\Sigma^b_{i,k}$-define all the $\Box^c_{i,k}$ functions.*

Proof. Let $f \in \Box^c_{i,k}$. Then f is computed by some U_{E^*}-uniform circuit family of genus k circuits with a fixed $\Sigma^p_{i-1,k}$ oracle X. The existence of a computation to construct and simulate this circuit, with correct oracle queries, is certainly $\Sigma^b_{i,k}$, but we must show that S^{i-1}_k can *prove* that existence. I'll describe the algorithm first, then formalize it in S^{i-1}_k. First, assume the circuits are treelike (which is OK for $\Box^c_{i,k}$) and levelled. We'll scan from left to right across the leaf layer, at each step computing the values of all the gates in the whole circuit whose descendant leaves are among the ones we've visited. For each leaf, this requires adding on at most the unique path from that leaf to the root, which is length $||\mathcal{L}_k||$, and we can find the lexicographically maximum (and hence correct) string of this length.

Let $Anc(r, x)$ be the number of gates in the circuit for x, all of whose descendant leaves are numbered r or lower. Define $InfoTuple(w, r, x)$ to be the Σ^b_{i-1} formula asserting that

- w is an r-tuple of bit strings of lengths $Anc(1, x)$, $Anc(2, x) - Anc(1, x), \ldots$, $Anc(r, x) - Anc(r - 1, x)$ respectively, and
- for all $j < |w_r|$, if the j-th most significant bit of w_r is 1, then the j-th ancestor gate of leaf r outputs 1 when given input as specified in w.

Thus *InfoTuple* asserts that w is a possible assignment of values to the ancestors of leaves $1, \ldots, r$, in which at least all the "yes" answers are correct for their presumed inputs.

Define $GtrTuple(w, w')$ to be a Δ^b_0 formula asserting that for some r, r',

- w and w' are tuples of r and r' elements respectively,
- $r \geq r'$,
- $(\forall j < r')(w_j \geq w'_j)$, and
- $(r > r') \vee (\exists j < r')(w_j > w'_j)$.

That is, each bit-string entry of w, treated as a binary number, is at least as great as the corresponding entry of w', and either at least one entry of w is *greater* than the corresponding entry of w' or w is longer than w'.

Now define $InfoSeq(W, r, x)$ to be the Σ^b_i formula asserting that

- W is a sequence of exactly r tuples,
- $(\forall j < r)\mathit{InfoTuple}(W_j, j, x)$, and
- $(\forall j < r-1)\mathit{GtrTuple}(W_{j+1}, W_j)$.

By LIND on $(\exists W, r)\mathit{InfoSeq}(W, r, x)$, S^i_k can prove there is a maximum-length W satisfying $\mathit{InfoSeq}(W, r, x)$ with r bounded by the number of leaves in the circuit; note that each tuple has $|\mathcal{L}_k|$ entries, each entry is bounded in numeric value by $|\mathcal{L}_k|$, and each tuple in such a sequence must have a distinct set of entries. Let w denote the last element of this maximum-length W. It must contain a bit-string for each leaf, or w (and therefore W) could be extended by adding on a string of zeroes for the next leaf. The first bit-string, w_1, must have the maximum possible value consistent with input x, or W could be extended by increasing w_1. Thus all the "no" answers in w_1 must be correct. Similarly, w_2 must have the maximum possible value consistent with x and w_1, and so on; after $|\mathcal{L}_k|$ steps we conclude that every gate value in w is correct. Extracting the value of the function is then straightforward. □

Together with the first part of Theorem 35, this implies Theorem 47. The second part of Theorem 35, together with Corollary 33, gives us some insight as to why it has been so difficult to strengthen Theorem 47 to $k = 2$:

Theorem 49. *For $i \geq 2$, if R^i_2 is conservative over S^{i-1}_2 with respect to $\forall\exists\Sigma^b_{i,2}$ sentences, then $\mathcal{FNC}^{\Sigma^p_{i-1,2}} = \square^c_{i,2}$, and hence $\mathcal{NC}^{\Sigma^p_{i-1,2}} = \Delta^c_{i,2}$.*

Since $\Delta^c_{i,2} = \left(\mathcal{NC}^1\right)^{\Sigma^p_{i-1,2}}$, this would imply a relativized collapse of $\mathcal{NC}$ down to $\mathcal{NC}^1$. Of course, Wilson showed such a relativized collapse in [26], but he constructed only a recursive oracle; it would be surprising to find such an oracle within the polynomial hierarchy. Conversely, if an oracle collapsing $\mathcal{FNC}$ to $\mathcal{FNC}^1$ *were* found within, say, Σ^p_i, it would imply conservativity between R^j_2 and S^{j-1}_2 for all $j > i$.

6 Links to Complexity Hierarchies

The authors of [17, 18] defined a "student-teacher game", a dialogue between an omniscient teacher and a computationally limited student assigned to find the best solution to some problem. Each time the student presents a suboptimal solution, the teacher replies with "no, that's not optimal; see, here's a better solution," and the student may then use this better solution in constructing the next solution. (In fact, the student may simply parrot it back *as* the next solution. This algorithm is called the *trivial student.*) With the aid of theorem 35 and this computational model, we can prove several results analogous to those of [18].

6.1 A Student-Teacher Witnessing Theorem for R^i_k

Theorem 50. *For $i \geq 1, k \geq 3$, and $\phi \in \exists\Pi^b_{i,k}$, if $S^i_k \vdash (\exists y)(\forall z)\phi(a,y,z)$, then there are $\square^c_{i+1,k}$ functions $f_1, \ldots f_j$ such that*

$$S^i_k \vdash \phi(a, f_1(a,b_1)) \vee \phi(a, f_2(a,b_1), b_2) \vee \ldots \vee \phi(a, f_j(a, b_1, \ldots b_{j-1}), b_j) \quad .$$

That is, there is a $\square^c_{i+1,k}$ student that finds a witness for $(\exists y)(\forall z)\phi(a,y,z)$ within some constant number r of rounds, provably in S^i_k.

The proof, which appears in detail in the journal version, resembles that of Theorem A of [18], with two main differences. First, in place of the equational theory PV_{i+1}, which contains a function symbol for every function definition in a Cobham-style algebra for $\mathcal{FP}^{\Sigma^p_i}$, we define an equational theory $CV_{i+1,k}$ containing a function symbol for every function definition in $A_{i+1,k}$. Second, we replace prefix induction by divide-and-conquer induction, and successor induction by prefix induction, throughout.

6.2 Collapsing Bounded Arithmetic

At this point, a previous version of this paper contained an adaptation of the argument of [18] that $T^i_2 = S^{i+1}_2 \Rightarrow \Sigma^p_{i+1} \subseteq \Delta^p_{i+1}/\text{poly}$. However, Buss [8] recently gave a simpler proof of a stronger result, so I'll adapt that technique instead. The main substantive difference between the following proof and Buss's is the replacement of prefix induction with divide and conquer induction; however, the proof technique is interesting enough to warrant another exposition.

Theorem 51. *For $i \geq 1$, $k \geq 3$, if $S^i_k = R^{i+1}_k$, then*

1. *S^i_k proves that every Σ^b_{i+1} formula $B(x)$ is equivalent to a formula of the form $C(x, A(x))$ where A is a $(\Sigma^b_{i+2} \cap \Pi^b_{i+2})$-defined function of S^i_k depending only on $|x|$, and $C \in \Pi^b_{i+1}$;*
2. *S^i_k proves $\Sigma^p_{i+1,k} \subseteq \Delta^p_{i+1,k}/poly$;*
3. *$S^i_k = S^{i+1}_k$;*
4. *$S^i_k = S_k$, which is therefore finitely axiomatized; and*
5. *for every bounded formula in $\mathcal{L}_k$, S^i_k proves it equivalent to a Boolean combination of $\Sigma^b_{i+2,k}$ formulæ.*

The idea, as in other proofs (e.g. [18, 8]) involving student-teacher games, is to play the omniscient teacher, giving away as little information as possible, and playing for long enough that the student *must* show some semblance of originality. Later, a *non*-omniscient teacher faced with a new problem can enlist the aid of this same demonstrably nontrivial student and solve the problem. But before we can apply this technique, we must define some preliminary notions.

Definition 52. Π_i^B is the class of quantified Boolean formulæ in prenex form containing i blocks of like quantifiers, starting with a $\forall$.

$TRU^i(\phi, w)$ is a formula stating that ϕ encodes a Π_i^B formula and w encodes a satisfying assignment of its free variables.

$SAT^i(\phi)$ is the formula $(\exists w \leq \phi) TRU^i(\phi, w)$.

Note that TRU^i and SAT^i are formulæ in the language of bounded arithmetic, while ϕ is a numeric *encoding* of a quantified Boolean formula (but we shall often identify it with that quantified Boolean formula, for convenience). Note also that we assume the encodings of ϕ and w are reasonably efficient, encodable and decodable in ALOGTIME, and that any assignment of the free variables in ϕ has an encoding numerically less than ϕ itself. We assume standard functions for encoding and decoding sequences, e.g. *SeqLen*(s) = the length of the sequence encoded by s.

It is well known that SAT^i is complete for $\Sigma_{i+1,2}^p$, and therefore for $\Sigma_{i+1,2}^b$ formulæ, under polynomial-time, many-one reductions. The techniques of the proof are not computationally demanding, and can be formalized in T_2^i, S_2^i, or even (with care) R_2^i, and the reductions can be weakened to $\mathcal{NC}$ or even ALOGTIME. Indeed, for $k \geq 2$, SAT^i is complete for $\Sigma_{i+1,k}^p$ and $\Sigma_{i+1,k}^b$ under $\Box_{0,k}^c$ many-one reductions; the larger growth rates are needed only during the reduction. And by standard techniques, TRU^i is Δ_{i+1}^b with respect to R_2^i.

The problem we shall set our student is, given a sequence of n encoded formulæ $\phi_1, \phi_2, \ldots, \phi_n$, to find the longest initial sequence $\phi_1, \phi_2, \ldots, \phi_m$ all of which are satisfiable, and produce witnesses $w_1, w_2, \ldots, w_m$ for them. (To avoid funny behaviour at 0, we give the student a witness w_1 for ϕ_1 for free.) However, we'll accept a witnessed initial sequence if it is *within a factor of 2* in length of the longest one. That is, the student is to realize the principle:

$$*(\forall \phi_1, \ldots, \phi_n \leq 2^{|a|})(\exists w_1, \ldots, w_m \leq \phi) \mathit{MaxSoln}(a, \phi, \mathbf{w})$$

where *MaxSoln*$(a, \phi, \mathbf{w})$ is defined by

$$\begin{aligned}
&\mathit{SeqLen}(\mathbf{w}) \leq \mathit{SeqLen}(\phi) \land \\
&\quad (\forall j \leq \mathit{SeqLen}(\phi)) \phi_j \leq 2^{|a|} \land (\forall j \leq \mathit{SeqLen}(\mathbf{w})) w_j \leq \phi_j \land \\
&\quad (\forall j \leq \mathit{SeqLen}(w)) TRU^i(\phi_j, w_j) \land \\
&\quad (\mathit{SeqLen}(\phi) \geq 2 \cdot \mathit{SeqLen}(w) \supset \exists j \leq 2 \cdot \mathit{SeqLen}(w)) \neg SAT^i(\phi_j).
\end{aligned}$$

Note that $\mathit{MaxSoln} \in \Pi_{i+1}^b$, so (*) is a $\forall\exists\Pi_{i+1}^b$ statement, and provable in R_k^{i+1} by maximizing $p \leq ||\phi||$ in the Σ_{i+1}^b formula

$$(\exists \mathbf{w} \leq \phi)(|\mathit{SeqLen}(\mathbf{w})| = p \land (\forall j \leq \mathit{SeqLen}(\mathbf{w})) TRU^i(\phi_j, w_j)) \quad .$$

If, as we shall assume for the rest of this theorem, $S_k^i = R_k^{i+1}$ and therefore S_k^i could also prove principle (*), then Theorem 50 would imply the existence of a $\Box_{i,k}^c$ student who witnesses it in some constant number r of rounds. If in a given round, the student proposes a sequence of witnesses $w_1, \ldots, w_m$ which is *not* "nearly maximal", the teacher is obliged to demonstrate that by providing witnesses for at least the first $2m$ formulæ. Thus $S_k^i = R_k^{i+1}$ implies there are functions $f_1, \ldots, f_r$, Σ_{i+1}^b-defined in S_k^i, such that, for any witnesses $\mathbf{w}$ for ϕ,

$$\begin{aligned}
&\mathit{MaxSoln}(a, \phi, f_1(a, \phi, w_1)) \vee \\
&\mathit{MaxSoln}(a, \phi, f_2(a, \phi, w_1, \ldots, w_{2m_1})) \vee \\
&\mathit{MaxSoln}(a, \phi, f_3(a, \phi, w_1, \ldots, w_{2m_2})) \vee \\
&\vdots \\
&\vee\ \mathit{MaxSoln}(a, \phi, f_r(a, \phi, w_1, \ldots, w_{2m_{r-1}}))
\end{aligned}$$

where $m_j = \mathit{SeqLen}(f_j(a, \phi, w_1, \ldots, w_{2m_{j-1}}))$.

Define an *original witness* to be any witness the student provides without first being given. The trivial student, the one who never provides an original witness, will produce $m_1 = 1$, $m_2 = 2$, $m_3 = 4, \ldots, m_r = 2^{r-1}$. So to coax an original witness from our student, it suffices to let $n \geq 2^r$ and make all the formulæ satisfiable. Define the formula $\mathit{FindsOrigBy}(a, m, \langle\phi_1, \ldots, \phi_s\rangle)$ to be

$$\begin{aligned}
&(\forall \mathrm{w} \leq \phi \leq 2^{|a|})((\forall j \leq \mathit{SeqLen}(\phi))\mathit{TRU}^i(\phi_j, w_j)) \supset \\
&\quad (\mathit{SeqLen}(f_1(a, \phi, \langle w_1\rangle)) > 1 \vee \\
&\quad\ \mathit{SeqLen}(f_2(a, \phi, \langle w_1, w_2\rangle)) > 2 \vee \\
&\quad\ \mathit{SeqLen}(f_3(a, \phi, \langle w_1, w_2, w_3, w_4\rangle)) > 4\ \vee \ldots \\
&\quad \vee\ \mathit{SeqLen}(f_m(a, \phi, \langle w_1, \ldots, w_{2^{m-1}}\rangle)) > 2^{m-1}))
\end{aligned}$$

This states that, no matter what witnesses are provided for the formulæ ϕ, the game will provide at least one original witness within the first m rounds, and therefore an original witness for one of the first 2^m formulæ. The assumption $S^i_k = R^{i+1}_k$ implies

$$S^i_k \vdash (\forall \phi_1, \ldots, \phi_{2^r} \leq 2^{|a|}) \left((\bigwedge_{j=1}^{2^r} \mathit{SAT}^i(\phi_j)) \supset \mathit{FindsOrigBy}(a, r, \phi) \right) \quad .$$

It is conceivable (assuming $\mathit{FindsOrigBy}(a, m, \phi)$) that, although the game is guaranteed to give an original witness for one of the first 2^m formulæ, the student uses the later formulæ to find it; for example, suppose the last formula in the sequence encodes a polynomial-time algorithm for SAT! The truth value of $\mathit{FindsOrigBy}(a, m, \phi)$ may therefore depend on all of ϕ, even though only the first 2^m ϕ's are actually treated as formulæ to be witnessed. We therefore distinguish the parts of ϕ that might be witnessed from the parts serving only as "advice" to the computation, and write, e.g, $\mathit{FindsOrigBy}(a, m, \langle\phi_1, \ldots, \phi_{2^m}\rangle, A)$.

Now consider the formula $\mathit{PreAdvice}(a, m, A)$ defined by

$$(\forall \phi_1, \ldots, \phi_{2^m} \leq 2^{|a|}) \left((\bigwedge_{j=1}^{2^m} \mathit{SAT}^i(\phi_j)) \supset \mathit{FindsOrigBy}(a, m, \phi, A) \right) \quad .$$

This states that, for any sequence of 2^m satisfiable formulæ ϕ, and any witnesses for those formulæ, the game with advice A will provide an original witness for one of ϕ.

Recall that

$$S_k^i \vdash (\forall \phi_1, \ldots, \phi_{2^r} \leq 2^{|a|}) \left((\bigwedge_{j=1}^{2^r} SAT^i(\phi_j)) \supset \mathit{FindsOrigBy}(a, r, \phi) \right) \quad .$$

Rephrasing this in terms of advice, $S_k^i \vdash \mathit{PreAdvice}(a, r, \langle\rangle)$. Since r is a constant, S_k^i proves without induction that there is a minimum $m \leq r$ such that $(\exists A)\mathit{PreAdvice}(a, m, A)$. Let $\mathit{Advice}(a, m, A)$ assert that m is such a minimum, and A is a corresponding advice string. Then $S_k^i \vdash (\exists m, A)\mathit{Advice}(a, m, A)$. Finally, given a formula $\psi \in \Sigma_i^B$ and "auxiliary" formulæ $\phi_1, \ldots, \phi_{2^{m-1}}$, let $\mathit{TestSeq}(\phi, \psi)$ be the sequence of length 2^m formed by concatenating ϕ with 2^{m-1} copies of ψ.

Now we are ready to prove the theorem. I claim that S_k^i proves

$$\begin{array}{l} (\mathit{Advice}(a, m, A) \wedge \psi \leq 2^{|a|}) \supset \\ \quad (SAT^i(\psi) \leftrightarrow (\forall \phi_1, \ldots, \phi_{2^{m-1}} \leq 2^{|a|}) \\ \quad\quad (\bigwedge_{j=1}^{2^{m-1}} SAT^i(\phi_j) \supset \mathit{FindsOrigBy}(a, m, \mathit{TestSeq}(\phi, \psi), A). \end{array}$$

Proof. Assume $\mathit{Advice}(a, m, A)$ and $\psi \leq 2^{|a|}$. If ψ is satisfiable, then for any satisfiable sequence $\phi_1, \ldots, \phi_{2^{m-1}}$, $\mathit{TestSeq}(\phi, \psi)$ is a sequence of 2^m satisfiable formulæ and so, by $\mathit{PreAdvice}(a, m, A)$, an original witness will be found for it.

On the other hand, if ψ is not satisfiable, then any original witness for $\mathit{TestSeq}(\phi, \psi)$ must actually witness one of the first 2^{m-1} formulæ ϕ, using ψ only as extra advice. Thus $\mathit{FindsOrigBy}(a, m, \mathit{TestSeq}(\phi, \psi), A)$ is equivalent to $\mathit{FindsOrigBy}(a, m-1, \phi, \langle A, \psi \rangle)$. However, by minimality of m, there is *no* advice A' for which $\mathit{PreAdvice}(a, m-1, A')$ holds. In particular, for the advice $A' = \langle A, \psi \rangle$, there must exist a satisfiable sequence $\phi_1, \ldots, \phi_{2^{m-1}}$ such that $\mathit{FindsOrigBy}(a, m-1, \phi, A')$ is false. □

Thus in the presence of the "advice" A, which depends only on a, the Σ_{i+1}^b-complete formula $SAT^i(\psi)$ is equivalent to the Π_{i+1}^b formula

$$(\forall \phi_1, \ldots, \phi_{2^{m-1}} \leq 2^{|a|}) \left(\bigwedge_{j=1}^{2^{m-1}} SAT^i(\phi_j) \supset \mathit{FindsOrigBy}(a, m, \mathit{TestSeq}(\phi, \psi), A) \right) .$$

This proves the first part of the theorem: if $B(x) \in \Sigma_{i+1}^b$, then by the completeness of SAT^i, there is a poly-time function f such that $B(x) \leftrightarrow SAT^i(f(x))$. We can think of the advice A as a (not necessarily single-valued) function of $a \approx 2^{|f(x)|}$, $\Sigma_{i+2}^p \cap \Pi_{i+2}^p$-defined in S_k^i, so we have $B(x) \leftrightarrow C(f(x), A(f(x)))$ where $C(\phi, A)$ is the above Π_{i+1}^b formula.

The second part follows immediately. In fact, since the reduction function f above can be made quasilinear [24], we get $\Sigma_{i+1,k}^p \subseteq \Delta_{i+1,k}^p/\text{quasilinear}$.

The third, fourth, and fifth parts follow much as in [8].

7 Conclusions

The results of this paper may be viewed as filling several holes in our understanding of theories of bounded arithmetic:

- easily-described computational characterizations of the $\forall\exists\Sigma_i^b$ consequences of R_k^i,
- the provable single-valuedness of witnessing functions for these consequences,
- easily-described computational characterizations of the $\forall\exists\Sigma_{i+1}^b$ consequences of S_k^i,
- simpler complexity-theoretic implications of $S_k^i \equiv T_k^i$ than previously known,
- complexity-theoretic implications of $R_k^i \equiv S_k^i$,
- complexity-theoretic implications if the known $\forall\exists\Sigma_{i+1}^b$-conservativity between S_k^i and R_k^{i+1} were extended down to $k = 2$, and
- complexity-theoretic implications of $S_k^i \equiv R_k^{i+1}$.

There remain, of course, a few much smaller but still nagging holes. One is our inability so far to prove analogues of Theorems 50 and 51 for $k = 2$, precisely the case of the greatest computational interest. Another remains whether R_2^{i+1} is indeed $\forall\exists\Sigma_{i+1}^b$-conservative over S_2^i.

Krajíček [16] shows that the equivalence $S_2^i \equiv T_2^i$, like the equivalence $S_2^{i+1} \equiv T_2^i$, would have startling complexity-theoretic implications: any problem solvable in $\mathcal{P}^{\Sigma_{i-1}^p}$ could be solved with only $O(\log)$ many queries to its oracle. (It is not known whether this would imply the collapse of the polynomial hierarchy.) The author has not yet found a natural complexity class corresponding to the Σ_{i+1}^b-definable functions of R_k^i. Such a class would provide stronger (and presumably still less plausible) implications of $R_k^i \equiv S_k^i$.

The larger questions not addressed in this paper remain stubbornly unsolved: *are* there any equivalences among the theories R_k^i, S_k^i, and T_k^i, *is* the theory $T_k = S_k = R_k$ finitely axiomatizable for any k, and *do* the corresponding complexity classes collapse?

Acknowledgments

I would like to thank Sam Buss, not only for introducing me to the subject, but for suggesting possible ways to strengthen my preliminary results and for pointing me to related work in the literature. Thanks also to Judy Goldsmith, who provided postdoc support for me while I did the initial work on this project.

References

1. William Allen. Arithmetizing uniform $\mathcal{NC}$. *Annals of Pure and Applied Logic*, 53(1):1–50, 1991. See also *Divide and Conquer as a Foundation of Arithmetic*, Ph.D. thesis, University of Hawaii at Manoa, 1988.
2. Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *computational complexity*, 2:97–110, Dec 1992.

3. Stephen Bloch. *Divide and Conquer in Parallel Complexity and Proof Theory.* PhD thesis, University of California, San Diego, 1992.
4. Stephen Bloch. Function-algebraic characterizations of log and polylog parallel time. *computational complexity*, 4(2):175–205, 1994. See also *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, 193–206, 1992.
5. Stephen Bloch. Parameter-free induction in bounded arithmetic. In preparation for submission, 1995.
6. Samuel R. Buss. *Bounded Arithmetic.* Number 3 in Studies in Proof Theory. Bibliopolis (Naples), 1986.
7. Samuel R. Buss. Axiomatizations and conservation results for theories of bounded arithmetic. In *Proceedings of a Workshop in Logic and Computation*, AMS Contemporary Mathematics, May 1987.
8. Samuel R. Buss. Relating the bounded arithmetic and polynomial time hierarchies. Manuscript, 1994.
9. Samuel R. Buss, Jan Krajíček, and Gaisi Takeuti. Provably total functions in bounded arithmetic theories R_3^i, U_2^i and V_2^i. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 116–161. Oxford University Press, 1993.
10. A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.
11. Peter Clote. A first order theory for the parallel complexity class $\mathcal{NC}$. Technical Report BCCS-8901, Boston College, 1989.
12. Peter Clote and Gaisi Takeuti. Bounded arithmetic for $\mathcal{NC}$, *ALogTIME*, $\mathcal{L}$ and $\mathcal{NL}$. *Annals of Pure and Applied Logic*, 56:73–117, 1992.
13. A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science II*, pages 24–30. North-Holland, 1965.
14. Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In Richard M. Karp, editor, *Complexity of Computations*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73. 1974.
15. Jay Hook. *A many-sorted approach to predicative mathematics.* PhD thesis, Princeton University, 1983.
16. Jan Krajíček. Fragments of bounded arithmetic and bounded query classes. *Transactions of the AMS*, 338(2):587–598, August 1993.
17. Jan Krajíček, Pavel Pudlák, and Jiři Sgall. Interactive computations of optimal solutions. In *Mathematical Foundations of Computer Science*, volume 452 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 1990.
18. J. Krajíček, P. Pudlák, and G. Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52:143–153, 1991.
19. Daniel Leivant. Subrecursion and lambda representation over free algebras. In Samuel Buss and Philip Scott, editors, *Feasible Mathematics, Perspectives in Computer Science*, pages 281–291. Birkhäuser, 1990.
20. Edward Nelson. *Predicative Arithmetic.* Princeton University Press, 1986.
21. Rohit J. Parikh. Existence and feasibility in arithmetic. *Journal of Symbolic Logic*, 36:494–508, 1971.
22. Alexander A. Razborov. Bounded arithmetic and lower bounds in Boolean complexity. Manuscript, 1994.
23. W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–383, 1981.

24. C.P. Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM*, 25:136–145, 1978.
25. Christopher B. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31:169–181, 1985.
26. Christopher B. Wilson. Relativized NC. *Math. Systems Theory*, 20:13–29, 1987.

Program Extraction from Classical Proofs

Ulrich Berger and Helmut Schwichtenberg

Mathematisches Institut der Universität München, D–80333 München, Germany

Abstract. Different methods for extracting a program from a classical proof are investigated. A direct method based on normalization and the wellknown negative translation combined with a realizability interpretation are compared and shown to yield equal results. Furthermore, the translation method is refined in order to obtain optimized programs. An analysis of the proof translation shows that in many cases only small parts of a classical proof need to be translated. Proofs extracted from such refined translations have simpler type and control structure. The effect of the refinements is demonstrated at two examples.

1 Introduction

It is well known that from a classical proof of $\forall x \exists y B(x,y)$, B quantifier–free, one can extract a program t such that $\forall x\, B(x,tx)$ holds. We discuss two possibilities to do this. 1. A direct method, which uses the classical proof and proof normalization directly as an algorithm. 2. A translation of the classical proof into an intuitionistic one from which via a realizability interpretation a program can be extracted. We show that both methods yield the same algorithm.

Furthermore we try to answer the question wether "programs from classical proofs" is a useful device practically. We apply the proof translation to a simple but informative example: We prove classically that w.r.t. an unbounded function $f\colon \mathbb{N} \to \mathbb{N}$ such that $f(0) = 0$ each n has a root m, i.e., $f(m) \leq n < f(m+1)$ holds. We translate the proof and extract a program root: $\mathbb{N} \to \mathbb{N}$ (depending on f) such that $f(\mathrm{root}(n)) \leq n < f(\mathrm{root}(n)+1)$ holds for all n. It's interesting that the classical proof is extremely easy and short (even if fully formalized); considerably shorter than the intuitionistic proof one would give intuitively. However the extracted program is unnecessarily complicated. We take this as a motivation to study refinements of the proof translation yielding simpler programs.

Program extraction can be messy for mainly two reasons: 1. A completely formalized proof, using the basic axioms of arithmetic only, will in general be extremely long. This can be remidied by introducing additional global assumptions which are of such a form that they do not spoil the extraction. 2. When translating a classical derivation into an intuitionistic one, each atomic formula P is replaced by $(P \to A) \to A$, where A is the existential formula we want to prove. Thus existential formulas are spread all over the derivation and therefore each subderivation gets computational content. This means that the extracted program will be at least as long and complicated as the proof. Furthermore one has to pay for the additional assumptions introduced in 1., since their translations have to be proved. In general, these proofs use case splittings which later show up in the program.

In this paper we propose a refined proof translation which does *not* replace *all* atoms P by $(P \to A) \to A$. By a simple syntactical analysis of the assumptions used in the derivation one can determine a set of *critical* atoms which suffice to be replaced in order to make the translation work. Applying this refined translation to our root example simplifies the resulting program drastically. A second example, concerned with a search in binary trees, shows a similar effect.

It would be interesting to see if our refined method can be applied successfully to larger examples too. An candidate might be the classical proof of Higman's Lemma [3], [7]. This proof has been translated and implemented in the Nuprl system by [6]. It is not known how the translated proof (which is extremely big) is related to the known constructive proofs of Higman's Lemma [8], [1]. A refined translation might help answering this.

2 Preliminaries

Our basic logical calculus is the $\wedge\!\to\!\forall$–fragment of minimal natural deduction for first order logic over simply typed lambda–terms. Classical and intuitionistic arithmetic and extensions thereof will be introduced via axioms.

Types are the ground types boole and nat, $\rho \times \sigma$ and $\rho \to \sigma$.

Terms are the terms of Gödel's system T, i.e. built from typed variables x^ρ, the constants $\mathrm{true}^{\mathrm{boole}}$, $\mathrm{false}^{\mathrm{boole}}$, 0^{nat}, $S^{\mathrm{nat}\to\mathrm{nat}}$, $R_{\mathrm{nat},\rho}^{\rho\to(\mathrm{nat}\to\rho\to\rho)\to\mathrm{nat}\to\rho}$ (recursion), $R_{\mathrm{boole},\rho}^{\rho\to\rho\to\mathrm{boole}\to\rho}$ (case analysis) by pairing $\langle r, s\rangle^{\rho\times\sigma}$, projection $\pi_i(r^{\rho_0\times\rho_1})^{\rho_i}$, abstraction $(\lambda x^\rho r^\sigma)^{\rho\to\sigma}$ and application $(r^{\rho\to\sigma}s^\rho)^\sigma$.

We have the usual conversions (writing $t+1$ for St)

$$\begin{aligned}
(\lambda x\, r)s &\to r[s/x],\\
\pi_i\langle r_0, r_1\rangle &\to r_i,\\
R_{\mathrm{nat},\rho}rs0 &\to r,\\
R_{\mathrm{nat},\rho}rs(t+1) &\to st(R_{\mathrm{nat},\rho}rst),\\
R_{\mathrm{boole},\rho}rs\,\mathrm{true} &\to r,\\
R_{\mathrm{boole},\rho}rs\,\mathrm{false} &\to s.
\end{aligned}$$

It is well known that each term reduces to a unique normal form w.r.t. these conversions (cf. [11]). To simplify the calculus we will identify terms with the same normal form. We will write $=$ for equality modulo normal forms and $\equiv$ for syntactical identity.

Atomic formulas are $\bot$ and $P(\vec{t})$ where P is a predicate symbol and $\vec{t}$ is a list of terms. The types ρ_i of the terms t_i are specified by the arity $(\vec{\rho})$ of P. We have at least one predicate symbol atom of arity (boole). The formula atom(t) will play the same role as usually $t = 0$ does. Frequently we will abreviate atom(t) by t. We assume that we have assigned to every predicate symbol $P \neq$ atom of arity $(\vec{\rho})$ a closed term t_P of type $\vec{\rho} \to \text{boole}$ defining the characteristic function of P. This means that we consider only decidable predicates.

Formulas are built from atomic formulas by conjunction $A \wedge B$, implication $A \to B$ and universal quantification $\forall x^\rho A$. Negation, disjunction and the existential quantifier are defined by

$$\begin{aligned} \neg A &:\equiv A \to \bot, \\ A \vee B &:\equiv \neg A \wedge \neg B \to \bot, \\ \exists x\, A &:\equiv \neg \forall x\, \neg A. \end{aligned}$$

By $\forall A$ we denote the universal closure of the formula A. A *Π-formula* is a formula of the form $\forall \vec{x}\, C$, where C is quantifier–free.

Axioms are divided into two groups.

Induction axioms:

$$\begin{aligned} \text{Ind}_{n,A}\colon\ & \forall . A[0/n] \to (\forall n. A \to A[n+1/n]) \to \forall n\, A, \\ \text{Ind}_{p,A}\colon\ & \forall . A[\text{true}/p] \to A[\text{false}/p] \to \forall p\, A. \end{aligned}$$

Π-axioms: All these axioms are closed *Π-formulas.*

$$\begin{aligned} \text{T}\colon\ & \text{atom}(\text{true}), \\ \neg\text{F}\colon\ & \neg\text{atom}(\text{false}), \end{aligned}$$

and for each predicate symbol P

$$\begin{aligned} \text{Efq}_P\colon\ & \forall \vec{x}. \bot \to P(\vec{x}), \\ \text{Stab}_P\colon\ & \forall \vec{x}. \neg\neg P(\vec{x}) \to P(\vec{x}), \\ \text{Dec}_{P,0}\colon\ & \forall \vec{x}. P(\vec{x}) \to \text{atom}(t_P \vec{x}), \\ \text{Dec}_{P,1}\colon\ & \forall \vec{x}. \text{atom}(t_P \vec{x}) \to P(\vec{x}). \end{aligned}$$

Clearly there is a lot of redundance in these axioms. However, since the choice of the axioms in a particular proof influences the extracted program, it is important to have a rich axiom system.

Derivations d^A are built from assumptions u^B and axioms by the introduction and elimination rules for $\wedge$, $\to$ and $\forall$:

$$\begin{array}{ll} (\lambda u^A\, d^B)^{A \to B}, & (d^{A \to B} e^A)^B, \\ \langle d^A, e^B \rangle^{A \wedge B}, & \pi_i(d^{A_0 \wedge A_1})^{A_i}, \\ (\lambda x^\rho\, d^A)^{\forall x^\rho A}, & (d^{\forall x^\rho A} t^\rho)^{A[t/x^\rho]} \end{array}$$

with the usual variable condition in the case of $(\lambda x\, d)^{\forall x A}$. We will write $\Gamma \vdash A$ if there is a derivation d^A with free assumptions among Γ. It's easy to see that we can derive stability $\neg\neg A \to A$ for all formulas A. Hence we have full classical arithmetic. Furthermore each quantifier–free formula A is decidable, i.e., there is a boolean term t_A such that $\vdash A \leftrightarrow \text{atom}(t_A)$. This can be used to do case splitting according to quantifier-free formulas A, i.e. for every formula B we can prove

$$\text{Cases}_{A,B}\colon (A \to B) \to (\neg A \to B) \to B.$$

The derivation $\mathrm{Cases}_{A,B}$ (which we will use later on) is given by

$$\lambda u_1, u_2.\mathrm{Ind}\,\vec{x}(\lambda u_3 \lambda u_4.u_3\mathrm{T})(\lambda u_5 \lambda u_6.u_6\neg\mathrm{F})t_A(\lambda u_7.u_1(d_0 u_7))(\lambda u_8.u_2(d_1 u_8))$$

where $d_0^{\mathrm{atom}(t_A)\to A}$, and $d_1^{\neg\mathrm{atom}(t_A)\to\neg A}$ are derivations which exist according to the decidability of A, and the axioms and assumption variables with indices are (writing t for $\mathrm{atom}(t)$)

$$\mathrm{Ind}_{p,(p\to B)\to(\neg p\to B)\to B},$$

$u_1^{A\to B}$, $u_2^{\neg A\to B}$, $u_3^{\mathrm{true}\to B}$, $u_4^{\neg\mathrm{true}\to B}$, $u_5^{\mathrm{false}\to B}$, $u_6^{\neg\mathrm{false}\to B}$, $u_7^{t_A}$ and $u_8^{\neg t_A}$.

Conversion for derivations is defined similary as for terms in the usual way.

$$\begin{aligned}
\pi_i(\langle d_0, d_1\rangle) &\to d_i,\\
(\lambda u\, d)^{A\to B} e &\to d[e/u],\\
(\lambda x\, d)^{\forall x\, A} t &\to d[t/x],\\
\mathrm{Ind}_{n,A}\vec{r}de0 &\to d,\\
\mathrm{Ind}_{n,A}\vec{r}de(t+1) &\to et(\mathrm{Ind}_{n,A}\vec{r}det)\\
\mathrm{Ind}_{p,A}\vec{r}de\,\mathrm{true} &\to d,\\
\mathrm{Ind}_{p,A}\vec{r}de\,\mathrm{false} &\to e.
\end{aligned}$$

Again it can be shown by standard methods — just as for Gödel's T — that any derivation term in arithmetic has a unique normal form w.r.t. these conversions.

Intuitionistic arithmetic is obtained by extending the calculus above by a *constructive* or *strong existential quantifier* written $\exists^*$ (as opposed to $\exists$ defined by $\neg\forall\neg$). We add axioms corresponding to the usual introduction and elimination rules for $\exists^*$.

$$\begin{aligned}
\exists^{*+}_{x,A}&:\ \ \forall\forall x.A \to \exists^* x\, A,\\
\exists^{*-}_{x,A,B}&:\ \ \forall.(\exists^* x^\rho\, A) \to (\forall x^\rho.A \to B) \to B.
\end{aligned}$$

Of course, in general, stability is underivable for formulas containing $\exists^*$. Constructive disjunction $\vee^*$ can be defined by

$$A \vee^* B :\equiv \exists^* p.(p = \mathrm{true} \to A) \wedge (p = \mathrm{false} \to B).$$

Program extraction from derivations with $\exists^*$ is done via a *modified realizability interpretation* due to Kreisel (cf. [4]). First we associate with each formula A (possibly containing $\exists^*$) a list of types $\tau(A)$ as follows.

$$\tau(R(\vec{t})) :\equiv \varepsilon,$$

where ε denotes the empty list, and if $\tau(A) = \vec{\rho}$ and $\tau(B) = \sigma_1, \ldots, \sigma_n$ we let

$$\begin{aligned}
\tau(A \to B) &:\equiv \vec{\rho} \to \sigma_1, \ldots, \vec{\rho} \to \sigma_n,\\
\tau(A \wedge B) &:\equiv \vec{\rho}, \vec{\sigma},\\
\tau(\forall x^\rho\, B) &:\equiv \rho \to \sigma_1, \ldots, \rho \to \sigma_n,\\
\tau(\exists^* x^\rho\, B) &:\equiv \rho, \vec{\sigma}.
\end{aligned}$$

Instead of $\vec{\rho} \to \sigma_1, \dots, \vec{\rho} \to \sigma_n$ we will sometimes write $\vec{\rho} \to \vec{\sigma}$. To give some examples, let n, m, k be of type nat. Then

$$\begin{aligned}
\tau(\forall n \exists^* m\, R(n,m)) &\equiv \mathrm{nat} \to \mathrm{nat},\\
\tau(\forall n \exists^* m \exists^* k\, R(n,m,k)) &\equiv (\mathrm{nat} \to \mathrm{nat}), (\mathrm{nat} \to \mathrm{nat}),\\
\tau(\forall n \exists^* m\, R(n,m) \to \exists^* k\, Q(k)) &\equiv (\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat}.
\end{aligned}$$

For every formula A (possibly containing $\exists^*$) and every list $\vec{r}$ of type $\tau(A)$ we define a formula $\vec{r}\,\mathrm{mr}\,A$ without $\exists^*$, to be read $\vec{r}$ *modified realizes* A.

$$\begin{aligned}
\varepsilon\,\mathrm{mr}\,R(\vec{t}) &:\equiv R(\vec{t}),\\
r_1, \dots, r_n\,\mathrm{mr}\,(A \to B) &:\equiv \forall \vec{x}. \vec{x}\,\mathrm{mr}\,A \to r_1\vec{x}, \dots, r_n\vec{x}\,\mathrm{mr}\,B,\\
\vec{r}, \vec{s}\,\mathrm{mr}\,(A \wedge B) &:\equiv \vec{r}\,\mathrm{mr}\,A \wedge \vec{s}\,\mathrm{mr}\,B,\\
r_1, \dots, r_n\,\mathrm{mr}\,\forall x^\rho\, B &:\equiv \forall x^\rho\, r_1 x, \dots, r_n x\,\mathrm{mr}\,B,\\
r, \vec{s}\,\mathrm{mr}\,\exists^* x^\rho\, B &:\equiv \vec{s}\,\mathrm{mr}\,B[r/x].
\end{aligned}$$

Note that if A does not contain $\exists^*$ then $\tau(A) \equiv \varepsilon$ and $\varepsilon\,\mathrm{mr}\,A \equiv A$.

Definition. Assume that to any assumption variable u^B we have assigned a list $\vec{x}_u^{\tau(B)} = x_{u_1}^{\rho_1}, \dots, x_{u_n}^{\rho_n}$ of distinct variables, where $\rho_1, \dots, \rho_n = \tau(B)$. Relative to this assignment we define for any derivation d^A its *extracted terms* $\mathrm{ets}(d^A)$, by induction on d^A. If $\tau(A) = \sigma_1, \dots, \sigma_k$, then $\mathrm{ets}(d^A)$ will be a list $r_1^{\sigma_1}, \dots, r_k^{\sigma_k}$.

$$\begin{aligned}
\mathrm{ets}(\mathrm{Ind}_{n,A}) &:\equiv \lambda \vec{x} R_{\mathrm{nat}, \tau(A)},\\
\mathrm{ets}(\mathrm{Ind}_{p,A}) &:\equiv \lambda \vec{x} R_{\mathrm{boole}, \tau(A)},\\
\mathrm{ets}(\mathit{\Pi}\text{–axiom}) &:\equiv \varepsilon,\\
\mathrm{ets}(u^A) &:\equiv \vec{x}_u^{\tau(A)},\\
\mathrm{ets}(\lambda u^A\, d^B) &:\equiv \lambda \vec{x}_u^{\tau(A)}\, \mathrm{ets}(d),\\
\mathrm{ets}(d^{A \to B} e^A) &:\equiv \mathrm{ets}(d)\mathrm{ets}(e),\\
\mathrm{ets}(\langle d^A, e^B \rangle) &:\equiv \mathrm{ets}(d^A), \mathrm{ets}(e^B),\\
\mathrm{ets}(\pi_0(d^{A \wedge B})) &:\equiv \text{the head of } \mathrm{ets}(d^{A \wedge B}) \text{ of same length as } \tau(A),\\
\mathrm{ets}(\pi_1(d^{A \wedge B})) &:\equiv \text{the tail of } \mathrm{ets}(d^{A \wedge B}) \text{ of same length as } \tau(B),\\
\mathrm{ets}(\lambda x^\rho\, d^A) &:\equiv \lambda x^\rho\, \mathrm{ets}(d),\\
\mathrm{ets}(d^{\forall x^\rho A} t^\rho) &:\equiv \mathrm{ets}(d)t,\\
\mathrm{ets}(\exists^{*+}_{x,A}) &:\equiv \lambda \vec{x} \lambda x \lambda \vec{y}. x, \vec{y},\\
\mathrm{ets}(\exists^{*-}_{x,A,B}) &:\equiv \lambda \vec{x} \lambda x \lambda \vec{y} \lambda z_1 \dots \lambda z_n . z_1 x \vec{y}, \dots, z_n x \vec{y}.
\end{aligned}$$

To be precise, for the extracted terms of the induction axioms we need simultaneous recursion operators in case $\tau(A)$ has length > 1. These can be defined easily. Furthermore note that if $\mathrm{ets}(d) = r_1, \dots, r_k$ and $\mathrm{ets}(e) = \vec{s}$, then $\mathrm{ets}(d)\mathrm{ets}(e) = r_1\vec{s}, \dots, r_k\vec{s}$ and $\lambda \vec{x}\, \mathrm{ets}(d) = \lambda \vec{x}\, r_1, \dots, \lambda \vec{x}\, r_k$. In the last clause the (omitted) types are

$$x^\rho, \quad (\vec{y})^{\tau(A)} \quad \text{and} \quad z_j^{\rho \to \tau(A) \to \sigma_j},$$

where $\tau(B) = \sigma_1, \ldots, \sigma_n$.

The following can be proved easily.

Lemma. $\mathrm{FV}(\mathrm{ets}(d)) \subseteq \mathrm{FV}(d) \cup \{\vec{x}_u^{\tau(A)} : u^A \in \mathrm{FA}(d)\}$.

Lemma. *We have* $\mathrm{ets}(d[t/x]) = \mathrm{ets}(d)[t/x]$ *and* $\mathrm{ets}(d[e/u]) = \mathrm{ets}(d)[\mathrm{ets}(e)/\vec{x}_u]$, *and if d and e have the same normal form then* $\mathrm{ets}(d)$ *and* $\mathrm{ets}(e)$ *have the same normal form too.*

Soundness Theorem. *Assume that to any assumption variable u^A we have assigned a list $\vec{x}_u^{\tau(A)}$ and a new assumption variable $\tilde{u}\colon \vec{x}_u^{\tau(A)} \operatorname{mr} A$. Relative to this assignment we can find for any derivation $d\colon A$ with $\mathrm{FA}(d) = \{u_1\colon A_1, \ldots, u_n\colon A_n\}$ a derivation*

$$\mu(d)\colon \mathrm{ets}(d) \operatorname{mr} A$$

with $\mathrm{FA}(\mu(d)) \subseteq \{\tilde{u}_1\colon \vec{x}_1 \operatorname{mr} A_1, \ldots, \tilde{u}_n\colon \vec{x}_n \operatorname{mr} A_n\}$.

Proof. Induction on d.

As an example we compute the extracted terms of the derivation $\mathrm{Cases}_{A,B}$ for case splitting.

$$\mathrm{ets}(\mathrm{Cases}_{A,B}) = \lambda \vec{y}, \vec{z}.\mathbf{if} t_A \vec{y}\vec{z}$$

where $\mathbf{if} :\equiv \vec{R}(\lambda \vec{y}_1, \vec{z}_1.\vec{y}_1)(\lambda \vec{y}_1, \vec{z}_1.\vec{z}_1)$ and $\vec{y}, \vec{z}, \vec{y}_1, \vec{z}_1$ are lists of variables of types $\vec{\rho} :\equiv \tau(B)$. Clearly

$$\mathbf{if}\, \mathrm{true}\, \vec{r}\vec{s} = \vec{r}, \quad \mathbf{if}\, \mathrm{false}\, \vec{r}\vec{s} = \vec{s}.$$

For better readability we use for $\mathbf{if} t_A \vec{r}\vec{s}$ the notation

$$\mathbf{if} \quad A \quad \mathbf{then} \quad \vec{r} \quad \mathbf{else} \quad \vec{s} \quad \mathbf{fi}.$$

3 Proof translation and the direct method

As is well known a proof of a ∀∃–theorem with a quantifier–free kernel — where ∃ is viewed as defined by ¬∀¬ — can be used as a program. We describe a "direct method" to use such a proof as a program (cf. [9]), and compare it with Gödel's negative translation followed by Harvey Friedman's A–translation (cf. [2]) and the program extraction described above.

3.1 The direct method

Assume we have a classical derivation $d\colon \forall \vec{x} \exists \vec{y}\, B[\vec{x}, \vec{y}]$, B quantifier–free, from closed Π–assumptions $\mathrm{FA}(d) = \{v_1\colon \forall \vec{x}_1\, C_1, \ldots, v_n\colon \forall \vec{x}_n\, C_n\}$. We describe an algorithm which, applied to closed terms $\vec{r}$, returns terms $\vec{s}$ such that $B[\vec{r}, \vec{s}]$ holds provided the assumptions $v_1, \ldots, v_n$ are true. The algorithm proceeds in three steps.

1. Instanciate d to $\vec{r}$. We get $d\vec{r}\colon \exists\vec{y}\, B[\vec{r},\vec{y}]$, i.e.,
$$d\vec{r}\colon (\forall\vec{y}.B[\vec{r},\vec{y}] \to \bot) \to \bot.$$
2. Apply $d\vec{r}$ to a fresh assumption variable $u\colon \forall\vec{y}.B[\vec{r},\vec{y}] \to \bot$. We get
$$d\vec{r}u\colon \bot.$$
3. Normalize $d\vec{r}u\colon \bot$. From its normal form $(d\vec{r}u)\!\downarrow$ read off the *first instance*
$$\vec{s} :\equiv |(d\vec{r}u)\!\downarrow|.$$

Below we describe how the first instance $|(d\vec{r}u)\!\downarrow|$ is obtained.

Clearly we may assume that the Π–assumptions $v_i\colon \forall\vec{x}_i\, C_i$ do not contain $\wedge$ and that the formula $B[\vec{r},\vec{y}]$ is of the form $\bigwedge\!\!\!\bigwedge_i B_i[\vec{y}]$ where the B_i do not contain $\wedge$. Therefore the fresh assumption $u\colon \forall\vec{y}.B[\vec{r},\vec{y}] \to \bot$ in step 2. may assumed to be $u\colon \forall\vec{y}.B_1[\vec{y}] \to \ldots \to B_m[\vec{y}] \to \bot$. Finally we may assume that d contains no free object variables. If it does, substitute arbitrary closed terms for them.

Let $d\colon \bot$ (corresponding to $(d\vec{r}u)\!\downarrow\colon \bot$ above) be a normal derivation with $\mathrm{FV}(d) = \emptyset$ of $\bot$ from assumptions
$$u\colon \forall\vec{y}.B_1[\vec{y}] \to \ldots \to B_m[\vec{y}] \to \bot,$$
$$v_1\colon \forall C_1, \ldots, v_n\colon \forall C_n$$
where $\forall\vec{y}.B_1[\vec{y}] \to \ldots \to B_m[\vec{y}] \to \bot$ is a false and $\forall C_1, \ldots, \forall C_n$ are true closed Π-formulas. We define a list $|d|$ of closed terms, called the *first instance* of d, such that $B_1[|d|], \ldots, B_m[|d|]$ are true. $|d|$ is defined by induction on d. Since d is normal and $\mathrm{FV}(d) = \emptyset$ it does not contain axioms (exept the truth axiom, which is a closed Π–formulas and hence may be assumed to be among the Π–assumptions $\forall C_i$). To see this recall that the normal form of any closed term of type nat is of the form $S(S(S\ldots(S0)\ldots))$ and of any closed term of type boole is either true or false; hence all induction axioms unfold. Therefore d is of the form
$$w\vec{s}d_1 \ldots d_k,$$
where $\vec{s}$ are closed terms and $d_1, \ldots, d_k$ are derivations of closed quantifier–free formulas. We distinguish two cases.

1. $d_1, \ldots, d_k$ derive only true formulas (which can be decided, since the formulas are quantifier–free and closed). Then w cannot be one of the v_i since all $\forall C_i$ are true. Hence $d = u\vec{s}d_1 \ldots d_k$ and the d_i derive $B_i[\vec{s}]$. So let $|d| :\equiv \vec{s}$.
2. There is a minimal i such that d_i derives a false formula, $A_1 \to \cdots \to A_m \to \bot$ say. Then $A_1 \ldots, A_m$ are true. Without loss of generality we may assume that $d_i = \lambda w_1^{A_1} \ldots \lambda w_m^{A_m}\, e$ where $e\colon \bot$ contains assumptions among
$$u\colon \forall\vec{y}.B_1[\vec{y}] \to \ldots \to B_m[\vec{y}] \to \bot,$$
$$v_1\colon \forall C_1, \ldots, v_n\colon \forall C_n,$$
$$w_1\colon A_1, \ldots, w_m\colon A_m.$$
Therefore we can recursively define $|d| :\equiv |e|$.

3.2 Proof translation

Now we describe the more traditional way to obtain a program from a classical proof of a formula $\exists y\, B$, B quantifier free, from (not necessarily closed) Π–assumptions. First, stability axioms are removed by applying Gödel's negative translation. We obtain an intuitionistic proof of $\exists x\, B$. Since intuitionistically $\neg\forall x \neg B$ is equivalent to $\neg\neg\exists^* x\, B$ we can use the fact that intuitionistic arithmetic is closed under Markov's rule and obtain an intuitionistic proof of $\exists^* x\, B$. For the latter step we use Friedman's A–translation from [2].

In principle the negative translation is not necessary since we could *prove* stability for all atomic formulas by case splitting. However, these proofs would introduce characteristic functions t_P which might lead to inefficient programs.

The negative translation replaces each atomic formula P by $(P \to \bot) \to \bot$ and the A–translation in turn replaces P by $P \vee^* A$, where $A :\equiv \exists^* x\, B$. So, at the end, P is replaced by $((P \vee^* A) \to A) \to A$ which is intuitionistically equivalent to $(P \to A) \to A$. Hence we merge the two steps and define the *A–translation* B^A of a formula B to be obtained by replacing any atomic subformula P of B by $(P \to A) \to A$ (including $P = \bot$; this is, of course, not optimal but convenient for comparison with the direct method). A similar translation for first order logic due to Leivant [5] is described in [10], 2.3., page 64, Theorem 3.20 (i).

Note that any derivation d of some formula B from assumptions $C_1, \ldots, C_n$ becomes after the A–translation a derivation of B^A from $C_1^A, , \ldots, C_n^A$. To see this recall that our logical rules are those of minimal logic and hence give no extra treatment to falsity. Also the axiom schemes (exept the truth axiom, the falsity axiom and $\mathrm{Efq}_{\mathrm{atom}}$, which will be viewed as Π–assumptions) remain instances of the same axiom scheme after the A–translation. E.g. boolean induction

$$B[\mathrm{true}/p] \to B[\mathrm{false}/p] \to \forall p\, B$$

is translated into

$$B^A[\mathrm{true}/p] \to B^A[\mathrm{false}/p] \to \forall p\, B^A,$$

which again is an instance of boolean induction.

Let us look at what happens with Π–assumptions under the A–translation. Again we may assume that all formulas considered do not contain $\wedge$.

Lemma 1 *For any quantifier-free formula C there is a derivation $d\colon C \to C^A$.*

Proof. Induction on C. Let $C \equiv B_1 \to \ldots \to B_m \to R$ with an atom R. We must derive

$$(\vec{B} \to R) \to \vec{B}^A \to (R \to A) \to A.$$

So assume

$$\tilde{u} : \vec{B} \to R,$$
$$\tilde{v}_i : B_i^A,$$
$$w : R \to A.$$

We must show A.

Case $u_i^- : \neg B_i$ for some i. Let $B_i \equiv \vec{C}_i \to P_i$ with atoms P_i. Then

$$\tilde{v}_i : \vec{C}_i^{\,A} \to (P_i \to A) \to A$$

and we have

$$e_{ij}[u_i^-] :\equiv \mathrm{Stab}_{C_{ij}} \lambda v^{\neg C_{ij}}.u_i^- \lambda \vec{u}^{\vec{C}_i}.\mathrm{Efq}_{P_i}(vu_j) : C_{ij},$$
$$e_i[u_i^-] :\equiv \lambda w_i^{P_i}.u_i^- \lambda \vec{u}^{\vec{C}_i} w_i : \neg P_i.$$

By IH we have $d_{C_{ij}} : C_{ij} \to C_{ij}^A$. Hence

$$\tilde{v}_i(d_{C_{i1}} e_{i1}) \dots (d_{C_{in_i}} e_{in_i})(\lambda w_i^{P_i}.\mathrm{Efq}_A(e_i w_i)) : A.$$

Case $u_i^+ : B_i$ for all i. Then

$$w(\tilde{u} u_1^+ \dots u_m^+) : A$$

The extracted terms for this derivation are

$$\begin{aligned} d^{\mathsf{ets}} \equiv \lambda \vec{x}_1, \dots, \vec{x}_m, \vec{z}.\ & \mathbf{if}\ \ \neg B_1\ \ \mathbf{then}\ \ \vec{x}_1 d^{\mathsf{ets}}_{C_{11}} \dots d^{\mathsf{ets}}_{C_{1n_1}} \vec{0}\ \ \mathbf{else} \\ & \dots \\ & \mathbf{if}\ \ \neg B_m\ \ \mathbf{then}\ \ \vec{x}_m d^{\mathsf{ets}}_{C_{m1}} \dots d^{\mathsf{ets}}_{C_{mn_m}} \vec{0}\ \ \mathbf{else} \\ & \vec{z}\ \ \mathbf{fi} \dots \mathbf{fi}, \end{aligned}$$

where $\vec{x}_i, \vec{z}$ are the lists of variables associated with $\tilde{v}_i : B_i^A, w$.

Here we have used case splitting formulas B_i.

Up to now, the formula A could have been arbitrarily chosen. If we want to use the A–translation to extract the computational content from a classical proof we have to choose a particular A involving the strong existential quantifier.

Lemma 2 *Let $B_i[\vec{x}, \vec{y}]$ be quantifier–free formulas without $\wedge$, and*

$$A[\vec{x}] :\equiv \exists^* \vec{y} \bigwedge_i B_i[\vec{x}, \vec{y}].$$

Then we can find a derivation of $(\forall \vec{y}.B_1[\vec{x}, \vec{y}] \to \dots \to B_m[\vec{x}, \vec{y}] \to \bot)^{A[\vec{x}]}$.

Proof. Let $\vec{y}$ be given and assume $\tilde{v}_i : \vec{B}_i^A$ and $w : \bot \to A$. We must show A.

Case $u_i^- : \neg B_i$ for some i. Let $B_i \equiv \vec{C}_i \to P_i$ with atoms P_i. Then

$$\tilde{v}_i : \vec{C}_i^{\,A} \to (P_i \to A) \to A$$

and we have

$$e_{ij}[u_i^-] :\equiv \mathrm{Stab}_{C_{ij}} \lambda v^{\neg C_{ij}}.u_i^- \lambda \vec{u}^{\vec{C}_i}.\mathrm{Efq}_{P_i}(vu_j) : C_{ij},$$
$$e_i[u_i^-] :\equiv \lambda w_i^{P_i}.u_i^- \lambda \vec{u}^{\vec{C}_i} w_i : \neg P_i.$$

Using $d_{C_{ij}}\colon C_{ij} \to C_{ij}^A$ from Lemma 1 we obtain

$$\tilde{v}_i(d_{C_{i1}}e_{i1})\dots(d_{C_{in_i}}e_{in_i})(\lambda w_i^{P_i}.\mathrm{Efq}_A(e_i w_i))\colon A.$$

Case $u_i^+\colon B_i$ for all i. Then

$$\exists^+\vec{y}\langle u_1^+,\dots,u_m^+\rangle\colon A$$

Note that the assumption $w\colon \bot \to A[\vec{x}]$ has not been used. The extracted terms for this derivation are

$$\begin{array}{ll} d^{\mathrm{ets}} \equiv \lambda\vec{y},\vec{x}_1,\dots,\vec{x}_m,\vec{z}. & \mathbf{if}\quad \neg B_1 \quad \mathbf{then}\quad \vec{x}_1 d^{\mathrm{ets}}_{C_{11}}\dots d^{\mathrm{ets}}_{C_{1n_1}}\vec{0}\quad \mathbf{else} \\ & \dots \\ & \mathbf{if}\quad \neg B_m \quad \mathbf{then}\quad \vec{x}_m d^{\mathrm{ets}}_{C_{m1}}\dots d^{\mathrm{ets}}_{C_{mn_m}}\vec{0}\quad \mathbf{else} \\ & \vec{y}\quad \mathbf{fi}\dots\mathbf{fi}, \end{array}$$

where $\vec{x}_i, \vec{z}$ are the lists of variables associated with $\tilde{v}_i\colon B_i^A, w$.

Theorem (Friedman). *For any derivation*

$$d[u\colon \forall\vec{y}.B_1[\vec{x},\vec{y}] \to \dots \to B_m[\vec{x},\vec{y}] \to \bot, v_1\colon \forall C_1,\dots,v_n\colon\forall C_n]\colon \bot$$

with B_i, C_j quantifier-free and without $\wedge$ we can find a derivation

$$d^{\mathrm{tr}}[v_1\colon\forall C_1,\dots,v_n\colon\forall C_n]\colon \exists^*\vec{y}\bigwedge_i B_i[\vec{x},\vec{y}].$$

Proof. Let $A[\vec{x}] :\equiv \exists^*\vec{y}\bigwedge_i B_i[\vec{x},\vec{y}]$ and consider the $A[\vec{x}]$–translation

$$d^{A[\vec{x}]}[u',v_1',\dots,v_n']\colon (\bot \to A[\vec{x}]) \to A[\vec{x}]$$

where

$$\begin{aligned} u'\colon\ & (\forall\vec{y}.B_1[\vec{x},\vec{y}] \to \dots \to B_m[\vec{x},\vec{y}] \to \bot)^{A[\vec{x}]}, \\ v_1'\colon\ & (\forall C_1)^{A[\vec{x}]},\dots v_n'\colon(\forall C_n)^{A[\vec{x}]} \end{aligned}$$

of d, obtained by just changing some formulas. By Lemma 1 we have

$$d_{v_i}[v_i\colon\forall C_i]\colon (\forall C_i)^{A[\vec{x}]}.$$

By Lemma 2 (now using the particular choice of $A[\vec{x}]$) the $A[\vec{x}]$–translation of the assumption u is provable without assumptions:

$$d_u\colon (\forall\vec{y}.B_1[\vec{x},\vec{y}] \to \dots \to B_m[\vec{x},\vec{y}] \to \bot)^{A[\vec{x}]}$$

Substituting $d_{v_i}[v_i\colon\forall C_i]$ for v_i' and d_u for u' in $d^{A[\vec{x}]}$ and applying the result to $\mathrm{Efq}_{A[\vec{x}]}$ we obtain

$$d^{\mathrm{tr}}[v_1\colon\forall C_1,\dots,v_n\colon\forall C_n]\colon \exists^*\vec{y}\bigwedge_i B_i[\vec{x},\vec{y}],$$

where

$$d^{\mathrm{tr}} \equiv d^{A[\vec{x}]}[d_u,d_{v_1},\dots,d_{v_n}]\mathrm{Efq}_{A[\vec{x}]}.$$

Having obtained a proof d^{tr} of an existential formula $\exists^*\vec{y}\bigwedge_i B_i[\vec{x},\vec{y}]$ we can then apply the general method of extracting terms to this proof. It yields

$$(d^{\mathrm{tr}})^{\mathrm{ets}} \equiv (d^{A[\vec{x}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}},d_{v_1}^{\mathrm{ets}},\dots,d_{v_n}^{\mathrm{ets}}]\vec{0}, \qquad (1)$$

since extracting terms commutes with substitution.

3.3 Comparison

We now prove that the value of the extracted terms when instanciated with a list $\vec{r}$ of closed terms is in fact the same as the result of the direct method.

So consider again the situation of Friedman's Theorem, i.e. a derivation

$$d[u\colon \forall\vec{y}.B_1[\vec{x},\vec{y}] \to \dots \to B_m[\vec{x},\vec{y}] \to \bot, v_1\colon\forall C_1,\dots,v_n\colon\forall C_n]\colon\bot$$

with B_i, C_j quantifier–free and without $\land$. We just observed that the program $(d^{\mathrm{tr}})^{\mathrm{ets}}$ extracted from the translated derivation has the form (1) above. Let us try to understand how this program works. First, $(d^{A[\vec{x}]})^{\mathrm{ets}}$ closely follows the structure of d. The reason is that $d^{A[\vec{x}]}$ differs from d only with respect to the formulas affixed, and when forming the extracted terms this affects only the types and the arities of the lists of object variables associated with assumption variables.

In order to comprehend $d_{v_i}^{\mathrm{ets}}$ and d_u^{ets} let us have a second look at the proofs of Lemma 1 and 2. First note that $d_{v_i}[v_i\colon\forall C_i]\colon(\forall C_i)^{A[\vec{x}]}$ is obtained from $d_i\colon C_i \to C_i^{A[\vec{x}]}$ constructed in the proof of Lemma 1 by

$$d_{v_i} \equiv \lambda\vec{y}_i.d_i(v_i\vec{y}_i).$$

Since v_i has type $\forall C_i$, which is a Harrop formula, we have $d_{v_i}^{\mathrm{ets}} \equiv \lambda\vec{y}_i\, d_i^{\mathrm{ets}}$. Now from the proof of Lemma 1 we obtain

$$\begin{array}{rlllll} d_i^{\mathrm{ets}} \equiv \lambda\vec{x}_1,\dots,\vec{x}_m,\vec{z}. & \mathbf{if} & \neg B_1 & \mathbf{then} & \vec{x}_1 d^{\mathrm{ets}}_{C_{11}}\dots d^{\mathrm{ets}}_{C_{1n_1}}\vec{0} & \mathbf{else} \\ & \dots & & & & \\ & \mathbf{if} & \neg B_m & \mathbf{then} & \vec{x}_m d^{\mathrm{ets}}_{C_{m1}}\dots d^{\mathrm{ets}}_{C_{mn_m}}\vec{0} & \mathbf{else} \\ & \vec{z} & \mathbf{fi}\dots\mathbf{fi}, & & & \end{array} \tag{2}$$

where $C_i \equiv B_1 \to \dots \to B_m \to R$ with $B_i \equiv \vec{C}_i \to P_i$ and $\vec{x}_i, \vec{z}$ are the lists of variables associated with $\tilde{v}_i, w$. Furthermore, $d^{\mathrm{ets}}_{C_{ij}}$ are the extracted terms of derivations $d_{C_{ij}}\colon C_{ij} \to C_{ij}^{A[\vec{x}]}$ constructed by previous applications of Lemma 1. Similarly

$$\begin{array}{rlllll} d_u^{\mathrm{ets}} \equiv \lambda\vec{y},\vec{x}_1,\dots,\vec{x}_m,\vec{z}. & \mathbf{if} & \neg B_1 & \mathbf{then} & \vec{x}_1 d^{\mathrm{ets}}_{C_{11}}\dots d^{\mathrm{ets}}_{C_{1n_1}}\vec{0} & \mathbf{else} \\ & \dots & & & & \\ & \mathbf{if} & \neg B_m & \mathbf{then} & \vec{x}_m d^{\mathrm{ets}}_{C_{m1}}\dots d^{\mathrm{ets}}_{C_{mn_m}}\vec{0} & \mathbf{else} \\ & \vec{y} & \mathbf{fi}\dots\mathbf{fi}, & & & \end{array} \tag{3}$$

where $B_i \equiv \vec{C}_i \to P_i$ and $\vec{x}_i, \vec{z}$ are the lists of variables associated with $\tilde{v}_i, w$. Furthermore, $d^{\mathrm{ets}}_{C_{ij}}$ are the extracted terms of derivations $d_{C_{ij}}\colon C_{ij} \to C_{ij}^{A[\vec{x}]}$ constructed by previous applications of Lemma 1.

This analysis makes it possible to prove that the value of the extracted terms when instanciated with a list $\vec{r}$ of closed terms is in fact the same as the result of the direct method to read off the first instance provided by the instanciated derivation

$$\tilde{d}[\tilde{u}\colon \forall\vec{y}.B_1[\vec{r},\vec{y}] \to \dots \to B_m[\vec{r},\vec{y}] \to \bot, v_1\colon\forall C_1,\dots,v_n\colon\forall C_n]\colon\bot$$

Below we will show the following

Claim. *For any normal derivation*

$$e[\bar{u}\colon \forall \vec{y}.B_1[\vec{r},\vec{y}] \to \ldots \to B_m[\vec{r},\vec{y}] \to \bot, v_1\colon \forall C_1, \ldots, v_n\colon \forall C_n]\colon \bot$$

with $\mathrm{FV}(e) = \emptyset$ *we have*

$$|e| = [\![(e^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]0]\!].$$

We then obtain that the instanciation of the extracted terms (1) with $\vec{r}$ for $\vec{x}$, i.e.

$$(d^{A[\vec{x}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}, d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]0[\vec{r}/\vec{x}] \equiv (d[\vec{r}/\vec{x}]^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]\vec{0}$$

has as its value the list of closed terms which is the first instance of the instanciated derivation $d[\vec{r}/\vec{x}]$, i.e. $|d[\vec{r}/\vec{x}]{\downarrow}|$. For by the claim we have

$$\begin{aligned} |d[\vec{r}/\vec{x}]{\downarrow}| &= [\![((d[\vec{r}/\vec{x}]{\downarrow})^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]\vec{0}]\!] \\ &= [\![((d[\vec{r}/\vec{x}]^{A[\vec{r}]})^{\mathrm{ets}}){\downarrow}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]\vec{0}]\!] \\ &= [\![(d[\vec{r}/\vec{x}]^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]\vec{0}]\!], \end{aligned}$$

since normalization commutes with $A[\vec{r}]$–translation and the formation of extracted terms.

It remains to prove the claim. We use induction on e. Since e is normal, it must be of the form $e = w\vec{s}e_1 \ldots e_k$ with $w \in \{\bar{u}, v_1, \ldots, v_n\}$.

Case 1. $e_1, \ldots, e_k$ derive only true formulas. Then $w = \bar{u}$, $k = m$ and the e_i derive $B_i[\vec{r},\vec{s}]$. By definition $|e| :\equiv \vec{s}$. Furthermore

$$\begin{aligned} &(e^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]\vec{0} \\ &= d_u^{\mathrm{ets}}[\vec{r}/\vec{x}]\vec{s}(e_1^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots] \ldots (e_m^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}/\vec{x}], d_{v_1}^{\mathrm{ets}}, \ldots]\vec{0} \\ &= \vec{s} \end{aligned}$$

by the form (3) of d_u^{ets}, since all $B_i[\vec{r},\vec{s}] = (\vec{B}_i \to P_i)[\vec{r},\vec{s}]$ are true.

Case 2. There is a minimal i such that e_i derives a false formula, $D_{i1}[\vec{s}] \to \cdots \to D_{in_i}[\vec{s}] \to \bot$ say. Then $D_{i1}[\vec{s}], \ldots, D_{in_i}[\vec{s}]$ are true. Without loss of generality we may assume that $e_i = \lambda w_1^{D_{i1}[\vec{s}]} \ldots \lambda w_m^{D_{in_i}[\vec{s}]} f$ where $f\colon \bot$ contains assumptions among

$$\begin{aligned} &\bar{u}\colon \forall \vec{y}.B_1[\vec{r},\vec{y}] \to \ldots \to B_m[\vec{r},\vec{y}] \to \bot, \\ &v_1\colon \forall C_1, \ldots, v_n\colon \forall C_n, \\ &w_1\colon D_{i1}, \ldots, w_m\colon D_{in_i}. \end{aligned}$$

Therefore by definition $|e| \equiv |f|$. Furthermore, using the notation $d_{w_j}[w_j]\colon D_{ij}^{A[\vec{r}]}$ for the derivation obtained by applying Lemma 1 to D_{ij}, we have

$$\begin{aligned} &(e^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]\vec{0} \\ &= \begin{cases} d_u^{\mathrm{ets}}[\vec{r}]\vec{s}(e_1^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots] \ldots (e_k^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots]\vec{0} & \text{if } w = \bar{u} \\ d_{v_i}^{\mathrm{ets}}\vec{s}(e_1^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots] \ldots (e_k^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots]\vec{0} & \text{if } w = v_i \end{cases} \\ &= (e_i^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}]d_{w_1}^{\mathrm{ets}} \ldots d_{w_{n_i}}^{\mathrm{ets}}\vec{0} \quad \text{by (3) and (2), respectively} \\ &= (f^{A[\vec{r}]})^{\mathrm{ets}}[d_u^{\mathrm{ets}}[\vec{r}], d_{v_1}^{\mathrm{ets}}, \ldots, d_{v_n}^{\mathrm{ets}}, d_{w_1}^{\mathrm{ets}}, \ldots, d_{w_{n_i}}^{\mathrm{ets}}]\vec{0}, \end{aligned}$$

so the claim follows from the IH.

4 Refinements

In applications it will be important to produce extracted terms with as few as possible case distinctions, and also that the case distinctions should be over as simple as possible boolean terms. The following example will show that such improvements are indeed necessary.

4.1 The root example

Let $f\colon \mathrm{nat} \to \mathrm{nat}$ be an unbounded function with $f(0) = 0$. Then we can prove

$$\forall n \exists m. f(m) \leq n < f(m+1).$$

If e.g. $f(m) = m^2$, then this formula expresses the existence of an integer square root $m := [\sqrt{n}]$ for any n. More formally we can prove

$$\forall n \exists m. \neg n < f(m) \wedge n < f(m+1) \tag{1}$$

from the assumptions

$$v_1\colon \forall n\, \neg n < f(0), \qquad v_2\colon \forall n\, n < f(g(n)).$$

Here $<^{\mathrm{nat} \to \mathrm{nat} \to \mathrm{boole}}$ is the characteristic function of the natural ordering of the natural numbers and $r < s$ denotes atom$(<rs)$. We expressed $f(m) \leq n$ by $\neg n < f(m)$ and $f(0) = 0$ by $\forall n\, \neg n < f(0)$ to keep the formal proof as simple as possible. In order to have Π-assumptions we had to express the unboundedness of f by a witnessing function g.

Now let us prove (1). Let n be given and assume

$$u\colon \forall m. \neg n < f(m) \to n < f(m+1) \to \bot.$$

We have to show $\bot$. From v_1 and u we inductively get $\forall m\, \neg n < f(m)$. For $m := g(n)$ this yields a contradiction to v_2.

The derivation term corresponding to this proof is

$$d :\equiv \mathrm{Ind}_{m, \neg n < f(m)} n (v_1 n) u (g(n)) (v_2 n)\colon \bot.$$

Now let

$$A :\equiv \exists^* m. \neg n < f(m) \wedge n < f(m+1).$$

The program extracted from d is

$$(d^{\mathrm{tr}})^{\mathrm{ets}} \equiv (d^A)^{\mathrm{ets}}[d_u^{\mathrm{ets}}, d_{v_1}^{\mathrm{ets}}, d_{v_2}^{\mathrm{ets}}]0\colon \mathrm{nat}.$$

$(d^A)^{\mathrm{ets}}$ is the same as d exept that $\mathrm{Ind}_{m, \neg n < f(m)}$ has to be replaced by

$$\lambda n\, R_{\mathrm{nat}, (\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat} \to \mathrm{nat}}$$

(since $\tau((\neg n < f(m))^A) = \tau(((n < f(m) \to A) \to A) \to (\bot \to A) \to A) =$ (nat → nat) → nat → nat), and the assumption variables u, v_1, v_2 in d have to be replaced by (unary lists of) object variables

$$\begin{gathered} x_u\colon \mathrm{nat} \to [(\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat} \to \mathrm{nat}] \to (\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat} \to \mathrm{nat}, \\ x_{v_1}\colon \mathrm{nat} \to (\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat} \to \mathrm{nat}, \\ x_{v_2}\colon \mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}. \end{gathered}$$

The subprograms $d_u^{\mathrm{ets}}, d_{v_1}^{\mathrm{ets}}, d_{v_2}^{\mathrm{ets}}$ (of the same types as x_u, x_{v_1}, x_{v_2}) are given by

$$\begin{aligned} d_u^{\mathrm{ets}} \equiv \lambda m, x_1, x_2, k.\ & \textbf{if}\quad n < f(m) \quad \textbf{then} \quad x_1 \mathrm{id} 0 \quad \textbf{else} \\ & \textbf{if}\quad \neg n < f(m+1) \quad \textbf{then} \quad x_2 0 \quad \textbf{else} \\ & m \quad \textbf{fifi}, \\ d_{v_1}^{\mathrm{ets}} \equiv \lambda n, x_1, k.\ & \textbf{if}\quad \neg n < f(0) \quad \textbf{then} \quad x_1 0 \quad \textbf{else} \\ & k \quad \textbf{fi}, \\ d_{v_2}^{\mathrm{ets}} \equiv \lambda n, k\ \ & k. \end{aligned}$$

Hence the normal form of $(d^{\mathrm{tr}})^{\mathrm{ets}}$ is

$$\begin{aligned} R\ & (\lambda x_1, k. \textbf{if}\quad \neg n < f(0) \quad \textbf{then} \quad x_1 0 \quad \textbf{else} \quad k \quad \textbf{fi}) \\ & (\lambda m, w_1, w_2, k. \\ & \qquad \textbf{if}\quad n < f(m) \quad \textbf{then} \quad w_1 \mathrm{id} 0 \quad \textbf{else} \\ & \qquad \textbf{if}\quad \neg n < f(m+1) \quad \textbf{then} \quad w_2 0 \quad \textbf{else} \\ & \qquad m \quad \textbf{fifi}) \\ & (g(n)) \\ & (\lambda k\, k) \\ & 0. \end{aligned}$$

Informally, $(d^{\mathrm{tr}})^{\mathrm{ets}} = H(g(n), \lambda k\, k, 0)$ where $H\colon \mathrm{nat} \to (\mathrm{nat} \to \mathrm{nat}) \to \mathrm{nat} \to \mathrm{nat}$ is such that

$$\begin{aligned} H(0, x_1, k) = {} & \textbf{if}\quad \neg n < f(0) \quad \textbf{then} \quad x_1 0 \quad \textbf{else} \quad k \quad \textbf{fi} \\ H(m+1, x_1, k) = {} & \textbf{if}\quad n < f(m) \quad \textbf{then} \quad H(m, \mathrm{id}, 0) \quad \textbf{else} \\ & \textbf{if}\quad \neg n < f(m+1) \quad \textbf{then} \quad x_1 0 \quad \textbf{else} \\ & m \quad \textbf{fifi}. \end{aligned}$$

This program is correct, but it is unnecessarily complicated. We will now describe a refined A–translation which will simplify the type of the auxiliary function H as well as its if–then–else structure. The type reduction will be achieved by *not* replacing *all* atoms P by $(P \to A) \to A$.

4.2 The refined translation

Let L be a set of formulas. In our applications L will consist of the quantifier-free kernels of the lemmas $\forall C_i$, and in addition of the formula $B_1[\vec{x},\vec{y}] \to \ldots \to B_m[\vec{x},\vec{y}] \to \bot$, if our goal formula is $\forall\vec{x}\exists\vec{y}\bigwedge\!\!\!\bigwedge_i B_i[\vec{x},\vec{y}]$.

The set of *L–critical* relation symbols is the smallest set satisfying the following condition.

> If $(\vec{C}_1 \to P_1) \to \ldots \to (\vec{C}_m \to P_m) \to R(\vec{t})$ is a positive subformula of an L–formula, and if for some i $P_i \equiv \bot$ or $P_i \equiv Q(\vec{s})$ for some L–critical relation symbol Q, then R is L–critical.

We now define an A–translation relative to L:

$$R(\vec{t})^A :\equiv \begin{cases} (R(\vec{t}) \to A) \to A, & \text{if } R \text{ is } L\text{–critical} \\ R(\vec{t}), & \text{otherwise} \end{cases}$$

$$\bot^A :\equiv A$$

$$(B \to C)^A :\equiv B^A \to C^A.$$

We will write $P \in \mathcal{C}_L$ if the atom P is of the form $R(\vec{t})$ for some L–critical R or $P \equiv \bot$.

A quantifier-free formula $B_1 \to \ldots \to B_m \to R$ will be called *L–relevant* if $R \in \mathcal{C}_L$.

Lemma 1* *For any $B \in \mathrm{Neg}(L)$ and any $C \in \mathrm{Pos}(L)$ we can find the following derivations.*

(i) $d_C\colon C \to C^A$. Let $C \equiv \vec{B} \to R$. Then in case $R \in \mathcal{C}_L$ we need g_{B_i} for B_i L–relevant and f_{B_j} for B_j L–irrelevant, and in case $R \notin \mathcal{C}_L$ we need $f_{\vec{B}}$.

(ii) $e_{C,D}\colon ((C \to D) \to A) \to C^A$ for C L–relevant. Let $C \equiv \vec{B} \to R$. Then we need Cases_R if $R \not\equiv \bot$, and also g_{B_i} or f_{B_i} depending on whether B_i is L–relevant or not.

(iii) $f_B\colon B^A \to B$ for B L–irrelevant. Let $B \equiv \vec{C} \to R$. Then we need $d_{\vec{C}}$.

(iv) $g_B\colon B^A \to (B \to A) \to A$ for B L–relevant. Let $B \equiv \vec{C} \to R$. Then we need some $e_{C_i,D}$ for C_i L–relevant, and Cases_{C_j} as well as d_{C_j} for C_j L–irrelevant.

Proof. Simultaneously by induction on the subformulas of L–formulas. In each case we will also calculate the extracted terms, i.e., we define by simultaneous structural recursion the terms d_C^{ets}, $e_{C,D}^{\mathrm{ets}}$, f_B^{ets} and g_B^{ets}.

(i) Let $C \equiv \vec{B} \to R$. *Case* $R \in \mathcal{C}_L$, $R \not\equiv \bot$. We must derive

$$(\vec{B} \to R) \to \vec{B}^A \to (R \to A) \to A.$$

So assume $u\colon \vec{B} \to R$, $v_i\colon B_i^A$ and $w\colon R \to A$. We must show A.

Let B_i for $i \in \{1,\ldots,k\}$ be relevant and B_j for $j \in \{k+1,\ldots,m\}$ be irrelevant. Assume $u_i\colon B_i$ and note that $f_j v_j\colon B_j$. Then from $u\colon B_1 \to \ldots \to B_k \to B_{k+1} \to \ldots \to B_m \to R$ and $w\colon R \to A$ we get A. Now cancel $u_k\colon B_k$, yielding $B_k \to A$. Using the IH $g_{B_k}: B_k^A \to (B_k \to A) \to A$ and the assumption B_k^A we get A. Repeating this procedure we finally cancel $u_1: B_1$, yielding $B_1 \to A$. Using the IH $g_{B_1}: B_1^A \to (B_1 \to A) \to A$ and the assumption B_1^A we get A, as required. The derivation term is

$$\begin{aligned} d_C \equiv\ & \lambda u, v_1, \dots, v_m, w. \\ & g_{B_1} v_1 \lambda u_1^{B_1}. \cdots g_{B_k} v_k \lambda u_k^{B_k}.w \left(u u_1 \dots u_k (f_{B_{k+1}} v_{k+1}) \dots (f_{B_m} v_m)\right) \end{aligned}$$

and we get

$$d_C^{\text{ets}} \equiv \lambda \vec{x}_1, \dots, \vec{x}_k, \vec{z}.\ g_{B_1}^{\text{ets}} \vec{x}_1 (g_{B_2}^{\text{ets}} \vec{x}_2 \dots (g_{B_k}^{\text{ets}} \vec{x}_k \vec{z}) \dots)$$

where $\vec{x}_i, \vec{z}$ are the lists of variables associated with v_i, w.

Case $R \equiv \bot$. Similarly, using Efq_A instead of $w: R \to A$. Then in d_C^{ets} we leave out $\lambda\vec{z}$ and replace the $\vec{z}$ in the kernel by $\vec{0}$.

Case $R \notin \mathcal{C}_L$. Then all B_i are irrelevant. We must derive

$$(\vec{B} \to R) \to \vec{B}^A \to R.$$

So assume $u: \vec{B} \to R$ and $\vec{B}^A$. Using the IH $f_{B_i}: B_i^A \to B_i$ we obtain $\vec{B}$ and hence R. In this case $d_C^{\text{ets}} \equiv \varepsilon$.

(ii) *Case* R with $R \in \mathcal{C}_L$, $R \not\equiv \bot$. Then $R^A \equiv (R \to A) \to A$, and we must derive

$$((R \to D) \to A) \to (R \to A) \to A.$$

So assume $u: (R \to D) \to A$ and $v: R \to A$. Then clearly A can be derived, using Cases_R and Efq_D. The derivation term is

$$e_{R,D} \equiv \lambda u, v.\text{Cases}_R v \lambda u_1^{\neg R}.u \lambda u_2^R.\text{Efq}_D(u u_1)$$

and we get

$$e_{R,D}^{\text{ets}} \equiv \lambda \vec{x}, \vec{y}.\ \textbf{if} \quad R \quad \textbf{then} \quad \vec{y} \quad \textbf{else} \quad \vec{x} \quad \textbf{fi}.$$

Case R with $R \equiv \bot$. Then $R^A \equiv A$, and we must derive

$$((\bot \to D) \to A) \to A.$$

But this clearly can be done using Efq_D, and we have $e_{\bot,D}^{\text{ets}} \equiv \lambda \vec{x}\, \vec{x}$.

Case $B \to C$. We must derive

$$[((B \to C) \to D) \to A] \to B^A \to C^A.$$

So assume $u: ((B \to C) \to D) \to A$ and $v: B^A$. We must show C^A. First note that we can derive $(C \to D) \to A$ from our assumptions, using the fact that by IH we can derive $B^A \to (B \to A) \to A$ (with g_B or f_B, depending on whether B is relevant or not). But then the claim follows, since by IH

$$e_{C,D}: ((C \to D) \to A) \to C^A.$$

(iii) Let $B \equiv \vec{C} \to R$. Since B is irrelevant we have $R \notin \mathcal{C}_L$. Then $(\vec{C} \to R)^A \equiv \vec{C}^A \to R$. We must derive

$$(\vec{C}^A \to R) \to \vec{C} \to R.$$

But this is easy, using the IH $d_{C_i}\colon C_i \to C_i^A$. Clearly $f_B^{\text{ets}} \equiv \varepsilon$.

(iv) *Case* R with $R \in \mathcal{C}_L$, $R \not\equiv \bot$. Then $R^A \equiv (R \to A) \to A$, and we must derive

$$((R \to A) \to A) \to (R \to A) \to A,$$

which is trivial. We have $g_R^{\text{ets}} \equiv \lambda \vec{x}\, \vec{x}$.

Case R with $R \equiv \bot$. Then $R^A \equiv A$, and we must derive

$$A \to (\bot \to A) \to A,$$

which again is trivial. We have $g_\bot^{\text{ets}} \equiv \lambda \vec{x}, \vec{y}\,.\,\vec{x}$.

Case $C \to B$. By assumption B is relevant. We must derive

$$(C^A \to B^A) \to ((C \to B) \to A) \to A. \tag{2}$$

So assume $u\colon C^A \to B^A$ and $v\colon (C \to B) \to A$.

We first consider the case where C is relevant. Then we have $e_{C,B}\colon ((C \to B) \to A) \to C^A$ by IH(ii), hence B^A (using v and u). By IH(iv) for the shorter formula B we know $g_B\colon B^A \to (B \to A) \to A$ and hence $(B \to A) \to A$. But $B \to A$ can easily be derived from our hypothesis $v\colon (C \to B) \to A$ and hence we obtain A, as required. The derivation term is

$$g_{C \to B} \equiv \lambda u, v.g_B(u(e_{C,B}v))\lambda u_1^B.v\lambda u_2^C u_1$$

and we get

$$g_{C \to B}^{\text{ets}} \equiv \lambda \vec{x}, \vec{y}.g_B^{\text{ets}}(\vec{x}(e_{C,B}^{\text{ets}}\vec{y}))\vec{y}.$$

We finally consider the case where C is irrelevant. The derivation now uses Cases_C; so it suffices to first derive (2) under the additional hypothesis C, and then derive (2) under the additional hypothesis $\neg C$.

So assume $u^+\colon C$. Since by IH(i) $d_C\colon C \to C^A$ we obtain C^A and hence B^A (using $u\colon C^A \to B^A$). By IH(iv) for the shorter formula B we know $g_B\colon B^A \to (B \to A) \to A$ and hence $(B \to A) \to A$. But $B \to A$ can easily be derived from our hypothesis $v\colon (C \to B) \to A$ and hence we obtain A, as required.

Now assume $u^-\colon \neg C$. Then by ex–falso–quodlibet we obtain $C \to B$ and hence A, using our hypothesis $v\colon (C \to B) \to A$.

The derivation term is

$$g_{C \to B} \equiv \lambda u, v.\text{Cases}_C(\lambda u^+.g_B(u(d_C u^+))\lambda u_1^B.v\lambda u_2^C u_1)(\lambda u^-.v\lambda u_3^C.\text{Efq}_B(u^- u_3))$$

and we get

$$g_{C \to B}^{\text{ets}} \equiv \lambda \vec{x}, \vec{y}.\ \textbf{if}\quad C\quad \textbf{then}\quad g_B^{\text{ets}}(\vec{x} d_C^{\text{ets}})\vec{y}\quad \textbf{else}\quad \vec{y}\quad \textbf{fi}.$$

Lemma 2* *Let B_i be quantifier–free formulas and $A :\equiv \exists^* \vec{y} \bigwedge\!\!\!\bigwedge_i B_i$. Then we can find a derivation of*

$$\forall \vec{y}. B_1^A \to \ldots \to B_m^A \to A$$

involving Cases_{B_i} *and* $d_{C_{ij}}\colon C_{ij} \to C_{ij}^A$ *for relevant* $B_i \equiv \vec{C}_i \to P_i$ *and* $f_{B_j}\colon B_j^A \to B_j$ *for irrelevant* B_j.

Proof. Let $\vec{y}$ be given and assume $v_1\colon B_1^A, \ldots, v_m\colon B_m^A$. We must show A. We may assume that B_i for $i \in \{1, \ldots, k\}$ be relevant and B_j for $j \in \{k+1, \ldots, m\}$ be irrelevant

Case $u_i^-\colon \neg B_i$ for some $i \in \{1, \ldots, k\}$. Let $B_i \equiv \vec{C}_i \to P_i$ with atoms P_i. Then

$$v_i\colon \vec{C_i}^A \to (P_i \to A) \to A$$

in case $P_i \not\equiv \bot$ and

$$v_i\colon \vec{C_i}^A \to A$$

in case $P_i \equiv \bot$. We have

$$e_{ij}[u_i^-]\colon\equiv \mathrm{Stab}_{C_{ij}} \lambda v^{\neg C_{ij}}.u_i^- \lambda \vec{u}^{\vec{C}_i}.\mathrm{Efq}_{P_i}(vu_j)\colon C_{ij},$$
$$e_i[u_i^-]\colon\equiv \lambda w_i^{P_i}.u_i^- \lambda \vec{u}^{\vec{C}_i} w_i\colon \neg P_i.$$

Using $d_{C_{ij}}\colon C_{ij} \to C_{ij}^A$ from Lemma 1* we obtain in case $P_i \not\equiv \bot$

$$v_i(d_{C_{i1}} e_{i1}) \ldots (d_{C_{in_i}} e_{in_i})(\lambda w_i^{P_i}.\mathrm{Efq}_A(e_i w_i))\colon A$$

and in case $P_i \equiv \bot$

$$v_i(d_{C_{i1}} e_{i1}) \ldots (d_{C_{in_i}} e_{in_i})\colon A.$$

Case $u_i^+\colon B_i$ for all $i \in \{1, \ldots, k\}$. Then

$$\exists^+ \vec{y} \langle u_1^+, \ldots, u_k^+, f_{B_{k+1}} v_{k+1}, \ldots, f_{B_m} v_m \rangle\colon A.$$

The extracted terms for this derivation are

$$\begin{aligned} d^{\mathrm{ets}} \equiv \lambda \vec{y}, \vec{x}_1, \ldots, \vec{x}_m.\quad & \mathbf{if} \quad \neg B_1 \quad \mathbf{then} \quad \vec{x}_1 d_{C_{11}}^{\mathrm{ets}} \ldots d_{C_{1n_1}}^{\mathrm{ets}} \vec{0} \quad \mathbf{else} \\ & \ldots \\ & \mathbf{if} \quad \neg B_k \quad \mathbf{then} \quad \vec{x}_k d_{C_{k1}}^{\mathrm{ets}} \ldots d_{C_{kn_k}}^{\mathrm{ets}} \vec{0} \quad \mathbf{else} \\ & \vec{y} \quad \mathbf{fi} \ldots \mathbf{fi}, \end{aligned}$$

where $\vec{x}_i$ is the list of variables associated with $v_i\colon B_i^A$.

This A–translation relative to L simplifies the extracted terms a lot. First the derivations of $C \to C^A$ are necessary only for those lemmas $\forall C$ where C involves L–critical relation symbols (for otherwise we have $C^A \equiv C$). For the other lemmas a derivation of $C \to C^A$ provided by Lemma 1* involves in most cases only very few case distinctions. Finally, in the derivation given by Lemma 2* of the A–translation of our false assumption $\bigwedge\!\!\!\bigwedge_i B_i^A \to A$ direct case distinctions are only necessary for the *relevant* B_i; for the other B_j we have a derivation of $B_j^A \to B_i$ by Lemma 1*.

4.3 Examples refined

Let us now come back to our initial example and study the effect of our refinements there. We can relativize the A–translation to the set L of formulas consisting of

$$\begin{gathered} n < f(0) \to \bot, \\ n < f(g(n)), \\ (n < f(m) \to \bot) \to n < f(m+1) \to \bot. \end{gathered}$$

Since no positive subformula of an L–formula is an implication with a $<$–atom as its conclusion there are no L–critical relation symbols. Hence only negations $\vec{B} \to \bot$ are L–relevant.

We now repeat our treatment of the root example, based on the A–translation relative to L and the refined Lemmas 1* and 2*. The derivation term corresponding to the informal proof is

$$d :\equiv \mathrm{Ind}_{m,\neg n<f(m)} n(v_1 n)u(g(n))(v_2 n) \colon \bot.$$

Now let

$$A :\equiv \exists^* m. \neg n < f(m) \wedge n < f(m+1).$$

The program extracted from d is

$$(d^{\mathrm{tr}})^{\mathrm{ets}} \equiv (d^A)^{\mathrm{ets}}[d_u^{\mathrm{ets}}, d_{v_1}^{\mathrm{ets}}, d_{v_2}^{\mathrm{ets}}] \colon \mathrm{nat}.$$

$(d^A)^{\mathrm{ets}}$ is the same as d exept that $\mathrm{Ind}_{m,\neg n<f(m)}$ has to be replaced by

$$\lambda n\, R_{\mathrm{nat},\mathrm{nat}}$$

(since $\tau((n < f(m) \to \bot)^A) = \tau(n < f(m) \to A) = \mathrm{nat}$), and the assumption variables u, v_1 in d have to be replaced by (unary lists of) object variables

$$\begin{aligned} &x_u \colon \mathrm{nat} \to \mathrm{nat} \to \mathrm{nat}, \\ &x_{v_1} \colon \mathrm{nat} \to \mathrm{nat}, \end{aligned}$$

whereas v_2 has to be replaced by the empty list. The subprograms $d_u^{\mathrm{ets}}, d_{v_1}^{\mathrm{ets}}$ (of the same types as x_u, x_{v_1}) are given by (cf. Lemmas 2* and 1*)

$$\begin{aligned} &d_u^{\mathrm{ets}} \equiv \lambda m, x.\, \textbf{if} \quad n < f(m) \quad \textbf{then} \quad x \quad \textbf{else} \quad m \quad \textbf{fi}, \\ &d_{v_1}^{\mathrm{ets}} \equiv \lambda n\, 0. \end{aligned}$$

Hence the normal form of $(d^{\mathrm{tr}})^{\mathrm{ets}}$ is

$$R0(\lambda m, x.\, \textbf{if} \quad n < f(m) \quad \textbf{then} \quad x \quad \textbf{else} \quad m \quad \textbf{fi})(g(n)).$$

Informally, $(d^{\mathrm{tr}})^{\mathrm{ets}} = h(g(n))$ where $h \colon \mathrm{nat} \to \mathrm{nat}$ is such that

$$\begin{aligned} &h(0) = 0, \\ &h(m+1) = \textbf{if} \quad n < f(m) \quad \textbf{then} \quad h(m) \quad \textbf{else} \quad m \quad \textbf{fi}. \end{aligned}$$

We conclude with a further example which is still simple but exploits the general results of section 4.2 more seriously.

Let p be a decidable property of finite binary trees. We consider the problem of computing from a tree x_0 with px_0 another tree y_0 with py_0 but minimal, i.e., for no proper subtree of y_0 property p holds. Let $\tilde{p}x$ mean that p holds for no proper subtree of x. $\tilde{p}$ can be defined inductively by

$$v_1 \colon \tilde{p}\varepsilon, \qquad v_2 \colon \forall x, y. \neg px \to \neg py \to \tilde{p}x \to \tilde{p}y \to \tilde{p}\langle x, y \rangle$$

where ε denotes the empty tree and $\langle x, y \rangle$ denotes the tree with immediate subtrees x and y. We assume further

$$v_3 \colon px_0$$

and try to prove $\exists y. \tilde{p}y \wedge py$, i.e., under the additional assumption

$$u \colon \forall y. \tilde{p}y \to py \to \bot$$

we have to prove $\bot$. Of course, in the proof we may use induction on trees, i.e., for any formula $B(x)$ the axiom

$$\mathrm{Ind}_{B(x)} \colon B(\varepsilon) \to (\forall x, y \,.\, B(x) \to B(y) \to B(\langle x, y \rangle)) \to \forall x \, B(x).$$

Informally, from u and v_2 we get $\forall x, y \,.\, \tilde{p}x \to \tilde{p}y \to \tilde{p}\langle x, y \rangle$ and hence, by induction, using v_1, $\forall x \, \tilde{p}x$. This clearly contradicts u and v_3.

From the proof term

$$d = ux_0(\mathrm{Ind}_{\tilde{p}} v_1 (\lambda x, y, w_1^{\tilde{p}x}, w_2^{\tilde{p}y} \,.\, v_2 xy(uxw_1)(uyw_2)w_1 w_2)x_0)v_3$$

we obtain the program (letting now w_i range over functions from trees to trees)

$$(d^{\mathrm{tr}})^{\mathrm{ets}} = d_u^{\mathrm{ets}} x_0 (R d_{v_1}^{\mathrm{ets}} (\lambda x, y, w_1, w_2 \,.\, d_{v_2}^{\mathrm{ets}} xy (d_u^{\mathrm{ets}} x w_1)(d_u^{\mathrm{ets}} y w_2) w_1 w_2) x_0) d_{v_3}^{\mathrm{ets}}$$

where d_u^{ets} and the $d_{v_i}^{\mathrm{ets}}$ still have to be calculated. Looking at our assumptions u and v_i it is clear that only $\tilde{p}$ is relevant. Hence, using the recipe given in the proof of Lemmas 1* and 2*, we obtain

$$d_u^{\mathrm{ets}} = \lambda x, w \,.\, wx \quad \text{and} \quad d_{v_1}^{\mathrm{ets}} = d_{v_3}^{\mathrm{ets}} = \mathrm{id}.$$

The calculation of $d_{v_2}^{\mathrm{ets}}$ is a little bit more involved. We have

$$d_{v_2}^{\mathrm{ets}} = \lambda z_1, z_2, w_1, w_2, z \,.\, g_{\neg px}^{\mathrm{ets}} z_1 (g_{\neg py}^{\mathrm{ets}} z_2 (g_{\tilde{p}x}^{\mathrm{ets}} w_1 (g_{\tilde{p}y}^{\mathrm{ets}} w_2 z)))$$

and

$$\begin{aligned} g_{\neg px}^{\mathrm{ets}} &= \lambda z_1, z_2 \,.\, \mathbf{if} \quad px \quad \mathbf{then} \quad z_1 \quad \mathbf{else} \quad z_2 \quad \mathbf{fi} \\ g_{\neg py}^{\mathrm{ets}} &= \lambda z_1, z_2 \,.\, \mathbf{if} \quad py \quad \mathbf{then} \quad z_1 \quad \mathbf{else} \quad z_2 \quad \mathbf{fi} \\ g_{\tilde{p}x}^{\mathrm{ets}} &= g_{\tilde{p}y}^{\mathrm{ets}} = \mathrm{id}. \end{aligned}$$

Hence

$$\begin{aligned} d_{v_2}^{\mathrm{ets}} &= \lambda z_1, z_2, w_1, w_2, z\,. \\ &\qquad \textbf{if} \quad px \quad \textbf{then} \quad z_1 \quad \textbf{else} \quad \textbf{if} \quad py \quad \textbf{then} \quad z_2 \quad \textbf{else} \quad w_1(w_2 z) \quad \textbf{fifi}. \end{aligned}$$

Therefore we get

$$\begin{aligned} (d^{\mathrm{tr}})^{\mathrm{ets}} = R \;\; & \mathrm{id} \\ & (\lambda x, y, w_1, w_2, z. \\ & \qquad \textbf{if} \quad px \quad \textbf{then} \quad w_1 x \quad \textbf{else} \\ & \qquad \textbf{if} \quad py \quad \textbf{then} \quad w_2 y \quad \textbf{else} \quad w_1(w_2 z) \quad \textbf{fifi}) \\ & x_0 \\ & x_0. \end{aligned}$$

This means $(d^{\mathrm{tr}})^{\mathrm{ets}} = g(x_0, x_0)$ where

$$\begin{aligned} g(\varepsilon, z) &= \; z, \\ g(\langle x, y\rangle, z) &= \; \textbf{if} \quad px \quad \textbf{then} \quad g(x,x) \quad \textbf{else} \\ & \quad\; \textbf{if} \quad py \quad \textbf{then} \quad g(y,y) \quad \textbf{else} \quad g(x, g(y,z)) \quad \textbf{fifi}. \end{aligned}$$

References

1. Coquand, T.: A proof of Higman's lemma by structural induction. Manuscript (1993)
2. Friedman, H.: Classically and intuitionistically provably recursive functions. In: Higher Set Theory, SLNCS **699** (1978) 21–28
3. Higman, G.: Ordering by divisibility in abstract algebras. Proc. London Math. Soc. **2** (1952) 236–366
4. Kreisel, G.: Interpretation of analysis by means of constructive functionals of finite types. In: Constructivity in Mathematics, North–Holland, (1959) 101–128
5. Leivant, D.: Syntactic translations and provably recursive functions. Journal of Symbolic Logic **50** (1985), 682–688
6. Murthy, C.: Extracting Constructive Content from Classical Proofs. PhD thesis. Technical Report, Nr. 90–1151, Dep. of Comp. Science, Cornell Univ, Ithaca, New York (1990)
7. Nash-Williams, C.: On well–quasi–ordering finite trees. Proc. Cambridge Phil. Soc. **59** (1963) 833–835
8. Schütte, K., Simpson, S.G: Ein in der reinen Zahlentheorie unbeweisbarer Satz über endliche Folgen natürlicher Zahlen. Arch. Math. Logik **25** (1985) 75–89
9. Schwichtenberg, H.: Proofs as programs. In: Proof Theory. A selection of papers from the Leeds Proof Theory Programme 1990, Cambridge University Press (1992) 81–113
10. Troelstra, A. S., van Dalen, D.: Constructivism in Mathematics Vol. 1. An Introduction. In: Studies in Logic and the Foundations of Mathematics, North–Holland **121** (1988)
11. Troelstra, A. S.: Metamathematical Investigations of Intuitionistic Arithmetic and Analysis. SLNM **344** (1973)

Computation Models and Function Algebras

(Extended Abstract)[1]

Peter Clote[2]

Department of Computer Science
Department of Computer Science, Boston College
Chestnut Hill, MA 02167 USA
email: clote@bcuxs1.bc.edu

1 Introduction

The modern digital computer, a force which has shaped the latter part of the 20-th century, can trace its origins back to work in mathematical logic concerning the formalization of concepts such as *proof* and *computable function*. Numerous examples support this assertion. For instance, in his development of the universal Turing machine, A.M. Turing seems to have been the first, along with J. von Neumann, to understand the potential of memory-stored programs executed by a universal computational device. Moreover, certain function classes and proof systems can be viewed as prototypes of programming languages: the Kleene μ-calculus and imperative programming (PASCAL, C); resolution (Gentzen sequent calculus) and logic programming (PROLOG); the Church-Kleene λ-calculus and functional programming (LISP); the Girard system **F** (polymorphic λ-calculus) and polymorphic functional programming (ML).

One recurring theme in recursion theory is that of a *function algebra* — i.e. a smallest class of functions containing certain initial functions and closed under certain operations. In 1925, as a technical tool in his claimed sketch proof of the continuum hypothesis, D. Hilbert [48] defined classes of higher type functionals by recursion. In 1928, W. Ackermann [1] furnished a proof that the diagonal function $\varphi_a(a,a)$ of Hilbert [48], a variant of the Ackermann function, is not primitive recursive. In 1931, K. Gödel [35] defined the primitive recursive functions, calling them "rekursive Funktionen", and used them to arithmetize logical syntax via Gödel numbers for his incompleteness theorem. Generalizing Ackermann's work, in 1936 R. Péter [72] defined and studied the k-fold recursive

[1] This is a preliminary, condensed version of an article to appear in the *Handbook of Recursion Theory*, ed. E. Griffor.

[2] Research partially supported by NSF CCR-9102896 and by US–Czechoslovak Science and Technology Program Grant 93 025.

functions. The same year saw the introduction of the fundamental concepts of Turing machine (A.M. Turing [88]), λ-calculus (A. Church [15]) and μ-recursive functions (S.C. Kleene [57]). By restricting the scheme of primitive recursion to allow only limited summations and limited products, the *elementary functions* were introduced in 1943 by L. Kalmár [55]. In 1953, A. Grzegorczyk [39] studied the classes $\mathcal{E}^k$ obtained by closing certain fast growing "diagonal" functions under composition and *bounded primitive recursion* or *bounded minimization.*

H. Scholz's 1952 [79] question concerning the characterization of *spectra* $\{n \in \mathbf{N} \; : (\exists \text{ model } M \text{ of } n \text{ elements})(M \models \phi)\}$ of first order sentences ϕ, shown in 1974 by N. Jones and A. Selman [54] to equal NTIME($2^{O(n)}$), was the starting point for J.H. Bennett's work [9] in 1962. Among other results, Bennett introduced the key notions of *positive extended rudimentary* and *extended rudimentary* (equivalent to the notions of nondeterministic polynomial time NP and the polynomial time hierarchy PH), characterized the spectra of sentences of higher type logic as exactly the Kalmár elementary sets, and proved that *rudimentary* coincides with Smullyan's notion of *constructive arithmetic* (those sets definable in the language $\{0, 1, +, \cdot, \leq\}$ of arithmetic by first order bounded quantifier formulas). Only much later in 1976 did C. Wrathall [92] connect these concepts to computer science by proving that the linear time hierarchy LTH coincides with rudimentary, hence constructive arithmetic, sets. In 1963 R. W. Ritchie [73] proved that Grzegorczyk's class $\mathcal{E}^2$ is the collection of functions computable in linear space on a Turing machine. In 1965, A. Cobham [26] characterized the polynomial time computable functions as the smallest function algebra closed under Bennett's scheme of *bounded recursion on notation.*[3] These arithmetization techniques led to a host of characterizations of computational complexity classes by machine-independent function algebras in the work of D. B. Thompson [86] in 1972 on polynomial space, of K. Wagner [89] in 1979 on general time complexity classes, and others. Function algebra characterizations of *parallel* complexity classes and of certain *boolean circuit* complexity classes were given more recently by the author [25, 23] and B. Allen [2]. Higher type analogues of certain characterizations were given in 1976 by K. Mehlhorn [65], in 1991 by S. Cook and B. Kapron [56, 29] for sequential computation, and in 1993 by the author, A. Ignjatovic, B. Kapron [20] for parallel computation. Though distinct, the arithmetization techniques of function algebras are related to those used in proving numerous results like *(i) NP* equals generalized first order spectra (R. Fagin [31]), *(ii)* characterization of complexity classes via finite models (program of *definitional complexity theory* investigated by R. Fagin [32], N. Immerman [52, 53], Y. Gurevich [40], and others).

From this preceding short historical overview, it clearly emerges that *function algebras* and *computation models* are intimately related as the software (class of programs) and hardware (machine model) counterparts of each other.

[3] According to [65], later in 1972 independently K. Weihrauch proved a similar characterization.

Historically, these notions are among the central concepts of recursion theory, proof theory and theoretical computer science. Perhaps this is the reason that led K. Gödel [36] in 1975 to claim that the most important open problem in recursion theory is the classification of all total recursive functions, presumably in a hierarchy of function algebras determined by admitting more and more complex operations. While much work characterizing ever larger subrecursive hierarchies has been done by Buchholz, Girard, Sacks, Schwichtenberg, Schütte, Takeuti, Wainer and others, in this paper we concentrate principally on subclasses of the primitive recursive functions and their relations to computational complexity.

Historically, Cobham's machine independent characterization of the polynomial time computable functions was the start of modern complexity theory, indicating a robust and mathematically interesting field. As outlined in section 5, current work on type 2 and higher type function algebras suggests directions for the extension of complexity theory to higher type computation.

2 Machine computation models

In this paper, we survey a selection of results which illustrate the arithmetization techniques used in characterizing certain computation models by function algebras. For reasons of space, proofs will not be given. Familiarity with basic complexity theory is assumed. In particular, we assume familiarity with the Turing machine model (TM), and its variants (the Turing machine with random access (TM_{ra}), nondeterministic Turing machine (NTM), Σ_k-Turing machine, alternating Turing machine (ATM). oracle Turing machine (OTM)), and with the boolean circuit model, and parallel random access machine model.

3 Turing machines

Definition 1 (Chandra, Kozen, Stockmeyer [14]) A Turing machine M with *random access* (TM_{ra}) is given by a finite set Q of states, an input tape having no tape head, k work tapes, an *index query* tape and an *index answer* tape. Except for the input tape, all other tapes have a tape head. M contains a distinguished input query state q_I, in which state M writes onto the leftmost cell of the index answer tape that symbol which appears in the k-th input tape cell, where $k = \sum_{i<m} k_i \cdot 2^i$ is the integer whose binary representation is given by the contents of the query index tape. Unlike the oracle Turing machine the query index tape is not automatically erased after making an input bit query. A logtime TM_{ra} runs in time $O(\log n)$, where n is the length of the input.

Convention 2 From now on, unless otherwise indicated, for any sublinear runtime $T(n) = o(n)$, the intended Turing machine model is TM_{ra}, while for runtimes $T(n) = \Omega(n)$, the intended Turing machine model is the conventional

TM. This convention applies to deterministic, nondeterministic, and alternating Turing machines. While it is a simple exercise to show that PTIME is the same class, regardless of model, it appears to be an open problem to determine the relationship between DTIME($T(n)$) on TM and TM$_{ra}$, for $T(n) = \Omega(n)$.

The classes DTIME($T(n)$), NTIME($T(n)$), DSPACE($S(n)$), NSPACE($S(n)$), PTIME, etc. are defined as usual.

Definition 3 Let ALOGTIME denote ATIME($O(\log n)$), and ALINTIME$_{ra}$ be ATIME($O(n)$) on a TM$_{ra}$. The *logtime hierarchy* LH [resp. the *linear time hierarchy* LTH, resp. the *polynomial time hierarchy* PH] is the collection of languages $L \subseteq \Sigma^*$, for which L is accepted by an ATM in time $O(\log n)$ [resp. $O(n)$, resp. $n^{O(1)}$] with at most $O(1)$ alternations. Σ_k-TIME($T(n)$) is the collection of languages accepted by an ATM in time $O(T(n))$ with at most k alternations, beginning with an existential state.

Definition 4 A function $f(x_1, \ldots, x_n)$ has *polynomial growth* resp. *linear growth* resp. *logarithmic growth* if

$$|f(x_1, \ldots, x_n)| = O(\max_{1 \leq i \leq n} |x_i|^k), \text{for some } k$$

resp.

$$|f(x_1, \ldots, x_n)| = O(\max_{1 \leq i \leq n} |x_i|)$$

resp.

$$|f(x_1, \ldots, x_n)| = O(\log^k(\max_{1 \leq i \leq n} |x_i|), \text{for some } k.$$

The *graph* G_f satisfies $G_f(\vec{x}, y)$ iff $f(\vec{x}) = y$. The *bitgraph* B_f satisfies $B_f(\vec{x}, i)$ iff the i-th bit of $f(x)$ is 1. If $\mathcal{C}$ is a complexity class, then $\mathcal{FC}$ [resp. *Lin*$\mathcal{FC}$ resp. *Log*$\mathcal{FC}$] is the class of functions of polynomial [resp. linear resp. logarithmic] growth whose bitgraph belongs to $\mathcal{C}$. In this paper, $\mathcal{GC}$ will abbreviate *Lin*$\mathcal{FC}$. The iteration $f^{(n)}(x)$ is defined by induction on n: $f^{(0)}(x) = x$, $f^{(n+1)}(x) = f(f^{(n)}(x))$. With this notation, the iteration $\log^{(n)} x$ should not be confused with the power $\log^n x = (\log x)^n$.

3.1 Concurrent random access machine

Emerging around 1976-77 from the work of Goldschlager [37, 38], Fortune-Wyllie [33], and Shiloach-Vishkin [84], the *parallel random access machine* (PRAM) provides an abstract model of parallel computer for algorithm development.

A *concurrent random access machine* CRAM has a sequence $R_0, R_1, \ldots$ of random access machines which operate in a synchronous fashion in parallel. Each R_i has its own local memory, an infinite collection of registers, each of which can hold an arbitrary non-negative integer. Global memory consists of an infinite collection of registers accessible to all processors, which are used for reading the input, processor message passing, and output. Global registers are designated $M^g_0, M^g_1, M^g_2, \ldots$, and local registers by $M_0, M_1, M_2, \ldots$ – local registers of processor P_i might be denoted $M_{i,0}, M_{i,1}, \ldots$. A global memory register can be read simultaneously by several processors (*concurrent read*, rather than *exclusive read*). In the case where more than one processor may attempt to write to the same global memory register, the lowest numbered processor succeeds (*priority resolution* of write conflict in this *concurrent write* rather than *exclusive write* model).

Instructions are as follows.

$$
\begin{array}{rcl}
M_{res} & = & \text{constant} \\
M_{res} & = & \text{processor number} \\
M_{res} & = & M_{op1} \\
M_{res} & = & M_{op1} + M_{op2} \\
M_{res} & = & M_{op1} \dot{-} M_{op2} \\
M_{res} & = & MSP(M_{op1}, M_{op2}) \\
M_{res} & = & LSP(M_{op1}, M_{op2}) \\
M_{res} & = & *M_{op1} \\
M_{res} & = & *M^g_{op1} \\
*M_{res} & = & M_{op1} \\
*M^g_{res} & = & M_{op1} \\
\textsc{goto} & \text{label} & \\
\textsc{goto} & \text{label} & \textsc{if } M_{op1} = M_{op2} \\
\textsc{goto} & \text{label} & \textsc{if } M_{op1} \le M_{op2} \\
\textsc{halt} & &
\end{array}
$$

Cutoff subtraction is defined by $x \dot{-} y = x - y$, provided that $x \geq y$, else 0. The shift operators MSP and LSP are defined by

- $\textsc{msp}(x, y) = \lfloor x/2^y \rfloor$, *provided* that $y < |x|$, otherwise 'B',
- $\textsc{lsp}(x, y) = x - 2^y \cdot (\lfloor x/2^y \rfloor)$, *provided* that $y \leq |x|$, otherwise 'B'.

The CRAM model is due to N. Immerman [53], though there slightly different conventions are made.

Instructions with '$*$' concern indirect addressing. The instruction $M_{res} = *M_{op1}$ assigns to local register M_{res} the contents of local register with address given by the value M_{op1}. Similarly, $M_{res} = *M^g_{op1}$ performs an indirect read from global memory into local memory.

3.2 Circuit families

A directed graph G is given by a set $V = \{1, \ldots, m\}$ of vertices (or nodes) and a set $E \subseteq V \times V$ of edges. A *circuit* C_n is a labeled, directed acyclic graph whose nodes of in-degree 0 are called *input* nodes and are labeled by one of $0, 1, x_1, \ldots, x_n$, and whose nodes v of in-degree $k > 0$ are called *gates* and are labeled by a k-place function $\ell(v)$ from a *basis* set of boolean functions. A circuit has a unique *output* node of out-degree 0.

Boolean circuits have basis $\wedge$, $\vee$, $\neg$, where $\wedge$, $\vee$ may have unbounded fan-in (as described below, the AC^k [resp. NC^k] model concerns unbounded fan-in [resp. fan-in 2] boolean circuits). A *threshold* gate $\mathrm{TH}_{k,n}$ outputs 1 if at least k of its n inputs is 1. A *modular counting* gate $\mathrm{MOD}_{k,n}$ outputs 1 if the sum of its n inputs is evenly divisible by k. A *parity* gate $\oplus_n$ outputs 1 if an even number of its n inputs is 1.

Definition 5 ([77], [3]) For $k \geq 0$, AC^k [resp. NC^k] is the class of languages in LOGTIME-uniform $\mathrm{SIZEDEPTH}(n^{O(1)}, O(\log^k n))$ over the boolean basis, where $\wedge,\vee$ have unbounded fan-in [resp. fan-in 2], and $\mathrm{NC} = \cup_k \mathrm{AC}^k = \cup_k \mathrm{NC}^k$. $\mathrm{ACC}(k)$ is the class of languages in LOGTIME-uniform $\mathrm{SIZEDEPTH}(n^{O(1)}, O(1))$ over the basis $\wedge$, $\vee$, $\neg$, $\mathrm{MOD}_{k,n}$, where $\wedge,\vee$ have unbounded fan-in, and $\mathrm{ACC} = \cup_{k\geq 2}\mathrm{ACC}(k)$. TC^0 is the class of languages in LOGTIME-uniform $\mathrm{SIZEDEPTH}(n^{O(1)}, O(1))$ over the basis $\mathrm{TH}_{k,n}$.

4 Some recursion schemes

Kleene's normal form theorem [57] states that for each recursive (partial) function f there is an index e for which $f(\vec{x}) = U(\mu y[T(e, \vec{x}, y) = 0])$, where T, U are primitive recursive. The proof relies on arithmetizing computations via Gödel numbers, a technique introduced in [35] by Gödel, and with which Turing computable functions can be shown equivalent to μ-recursive functions. Since then, there have been a number of *arithmetizations* of machine models [57, 58, 9, 26, 73, 86, 89, 25, 23], etc. Key to all of these results is the availability in a function algebra $\mathcal{F}$ of a conditional function, a pairing function, and some string manipulating functions, in order to show that the function $\mathrm{NEXT}_M(x, c) = d$ belongs to $\mathcal{F}$, where c, d encode configurations of machine M on input x and d is the configuration obtained in one step from configuration c.

Definition 6 If $\mathcal{X}$ is a set of functions and OP is a collection of operations, then $[\mathcal{X}; \mathrm{OP}]$ denotes the smallest set of functions containing $\mathcal{X}$ and closed under the operations of OP. The set $[\mathcal{X}; \mathrm{OP}]$ is called a *function algebra.* If $\mathcal{F}$ is a class of functions, then $\mathcal{F}_*$ is the class of predicates whose characteristic function belongs to $\mathcal{F}$.

Definition 7 Let $\mathcal{F} = [f_1, f_2, \ldots; O_1, O_2, \ldots]$ be a function algebra. Let O denote operator O_{i_0}, and fix a derivation D of $f \in \mathcal{F}$. The rank $rk_{O,D}(f)$ of applications of O in the derivation D of $f \in \mathcal{F}$ is defined by induction. If f is an initial function $f_1, f_2, \ldots$ then $rk_{O,D}(f) = 0$. Suppose that f is defined by application of operator O_i to functions $g_1, \ldots, g_m$ where $rk_{O,D}(g_j) = r_j$ for $1 \leq j \leq m$. If $i = i_0$ then $rk_{O,D}(f) = 1 + \max\{r_1, \ldots, r_m\}$; otherwise $rk_{O,D}(f) = \max\{r_1, \ldots, r_m\}$. The *O-rank* $rk_O(f)$ of a function $f \in \mathcal{F}$ is the minimum of $rk_{O,D}(f)$ over all derivations D of f in $\mathcal{F}$.

Operations which have been studied in the literature include composition, primitive recursion, minimization, and their variants including bounded composition, bounded recursion, bounded recursion on notation, bounded minimization, simultaneous recursion, multiple recursion, course-of-values recursion, divide and conquer recursion, safe and tiered recursion, etc.[4] Good surveys of function algebras include the monographs by H. Rose [75] and K. Wagner and G. Wechsung [90] (chapters 2, 10).

4.1 An algebra for the logtime hierarchy LH

Definition 8 The *successor* function $s(x) = x + 1$; the *binary successor* functions s_0, s_1 satisfy $s_0(x) = 2 \cdot x$, $s_1(x) = 2 \cdot x + 1$; the n-place projection functions $I_k^n(x_1, \ldots, x_n) = x_k$; I denotes the collection of all projection functions.

Definition 9 The function f is defined by *composition* (COMP) from functions $h, g_1, \ldots, g_m$ if

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

The function f is defined by *primitive recursion* (PR) from functions g, h if

$$\begin{array}{rcl} f(0, \vec{y}) & = & g(\vec{y}), \\ f(x+1, \vec{y}) & = & h(x, \vec{y}, f(x, \vec{y})). \end{array}$$

The collection $\mathcal{PR}$ of primitive recursive functions is $[0, I, s; \text{COMP}, \text{PR}]$.

Primitive recursion defines $f(x+1)$ in terms of $f(x)$, so that the computation of $f(x)$ requires approximately $2^{|x|}$ many steps, an exponential number in the length of x. To define smaller complexity classes of functions, Bennett [9] introduced the scheme of *recursion on notation*, which Cobham [26] later used to characterize the polynomial time computable functions.

Definition 10 Assume that $h_0(x, \vec{y}), h_1(x, \vec{y}) \leq 1$. The function f is defined by *concatenation recursion on notation* (CRN) from g, h_0, h_1 if

$$\begin{array}{ll} f(0, \vec{y}) = g(\vec{y}) & \\ f(s_0(x), \vec{y}) = s_{h_0(x,\vec{y})}(f(x, \vec{y})), & \text{if } x \neq 0 \\ f(s_1(x), \vec{y}) = s_{h_1(x,\vec{y})}(f(x, \vec{y})). & \end{array}$$

[4] In this paper, for uniformity of notation, a number of operations are introduced as *bounded* instead of *limited* operations. For example, Grzegorczyk's schemes of *limited recursion* and *limited minimization* are here called *bounded recursion* and *bounded minimization*.

This scheme can be written in the abbreviated form

$$f(0,\vec{y}) = g(\vec{y})$$
$$f(s_i(x),\vec{y}) = s_{h_i(x,\vec{y})}(f(x,\vec{y})).$$

The scheme CRN was introduced by the author in [25] though motivated by a similar scheme due to J. Lind [64]. If concatenation of the empty string is allowed, or if the condition $h_i(x,\vec{y}) \leq 1$ is dropped (as in Lind's scheme), then the resulting scheme is provably stronger.

Definition 11 The length of x in binary satisfies $|x| = \lceil log_2(x+1)\rceil$; $\text{MOD}2(x) = x - 2 \cdot \lfloor \frac{x}{2} \rfloor$; the function $\text{BIT}(i,x) = \text{MOD}2(\lfloor \frac{x}{2^i} \rfloor)$ yields the coefficient of 2^i in the binary representation of x; the *smash* function satisfies $x\#y = 2^{|x|\cdot|y|}$. The functions MSP, LSP, *msp*, *lsp* satisfy $\text{MSP}(x,y) = \lfloor \frac{x}{2^y} \rfloor$ $\text{LSP}(x,y) = x \bmod 2^y$, $msp(x,y) = \lfloor x/2^{|y|} \rfloor$, $lsp(x,y) = x \bmod 2^{|y|}$. The algebra A_0 is defined to be

$$[0, I, s_0, s_1, \text{BIT}, |x|, \#; \text{COMP}, \text{CRN}].$$

Small sequences of small numbers can be encoded by a function in A_0.

Proposition 12 (Clote [17]) *If $f \in A_0$ then there exists $g \in A_0$ such that for all x,*

$$g(x,\vec{y}) = \langle f(0,\vec{y}), \ldots, f(|x|-1,\vec{y}) \rangle.$$

The following lemma, together with the sequence encoding machinery of A_0, allow a proof of $A_0 = \mathcal{F}\text{LH}$.

Lemma 13 *For every $k, m > 1$,*

$$\text{DTIMESPACE}(log^k(n), log^{1-1/m}(n)) \subseteq A_0.$$

Moreover, $\text{DSPACE}(O(\log\log(n)))$ *on a* TM_{ra} *is contained in* LH.

Theorem 14 (P. Clote) $A_0 = \mathcal{F}\text{LH}$.

Theorem 15 (Clote-Takeuti [23])

$$\text{TC}^0 \quad = \quad [0, I, s_0, s_1, |x|, \text{BIT}, \times, \#; \text{COMP}, \text{CRN}].$$

Remark 16 Theorem 14 was first obtained by combining the author's result [25] that A_0 equals FO definable functions, and the Barrington-Immerman-Straubing result [3] that FO = LH, an analogue of Bennett's Theorem 41. The current proof is direct, influenced by A. Woods' presentation in [91], and simplifies the argument of [3] by using Lemma 13.

4.2 Bounded recursion on notation

Definition 17 The function f is defined by *bounded recursion on notation* (BRN) from g, h_0, h_1, b if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}), \\ f(s_0(x),\vec{y}) &= h_0(x,\vec{y},f(x,\vec{y})) \text{ if } x \neq 0 \ , \\ f(s_1(x),\vec{y}) &= h_1(x,\vec{y},f(x,\vec{y})). \end{aligned}$$

Theorem 18 (A. Cobham [26], see H. Rose [75])

$$\mathcal{F}\text{PTIME} = [0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}].$$

Using the same techniques, one can characterize the class $\mathcal{G}$P polynomial time computable functions of linear growth as follows. (recall that $*$ is concatenation), and the other assertion follows by alternate functions in bounding the recursion on notation.

Theorem 19 (D.H. Thompson [86])

$$\begin{aligned} \mathcal{G}\text{PTIME} &= [0, I, s_0, s_1, *; \text{COMP}, \text{BRN}] \\ &= [0, I, s_0, s_1, \times; \text{COMP}, \text{BRN}]. \end{aligned}$$

Definition 20 The function f is defined from functions g, h_0, h_1, k by *sharply bounded recursion on notation*[5] (SBRN) if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(s_0(x),\vec{y}) &= h_0(x,\vec{y},f(x,\vec{y})), \text{ if } x \neq 0 \\ f(s_1(x),\vec{y}) &= h_1(x,\vec{y},f(x,\vec{y})), \end{aligned}$$

provided that $f(x,\vec{y}) \leq |k(x,\vec{y})|$ for all $x, \vec{y}$.

In [64], J. Lind characterized $\mathcal{F}$LOGSPACE functions on words $w \in \Sigma^*$ as the smallest class of functions containing the initial functions $c_=$ (characteristic function of equality), $*$ (string concatenation) and closed under the operations of explicit transformation, log bounded recursion on notation, and a (provably stronger) version of concatenation on notation. An arithmetic version of Lind's characterization is the following.

Theorem 21

$$\begin{aligned} \mathcal{F}\text{LOGSPACE} &= [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{SBRN}] \\ &= [0, I, s_0, s_1, msp, \#; \text{COMP}, \text{CRN}, \text{SBRN}]. \end{aligned}$$

[5] In [23], this scheme was denoted B_2RN.

The first statement appeared in [24, 23] and the second can be proved using similar techniques.

Recently, function algebras have been found for small parallel complexity classes. Consider the following variants of recursion on notation.

Definition 22 The function f is defined by *k-bounded recursion on notation* ($k - \text{BRN}$) from g, h_0, h_1 if

$$\begin{array}{rcl} f(0,\vec{x}) & = & g(\vec{x}) \\ f(2n,\vec{x}) & = & h_0(n,\vec{x},f(n,\vec{x})), \text{ if } n \neq 0 \\ f(2n+1,\vec{x}) & = & h_1(n,\vec{x},f(n,\vec{x})) \end{array}$$

provided that $f(n,\vec{x}) \leq k$ holds for all $n, \vec{x}$.

Definition 23 The function f is defined by *weak bounded recursion on notation* (WBRN) from g, h, k if

$$\begin{array}{rcl} F(0,\vec{x}) & = & g(\vec{x}) \\ F(2n,\vec{x}) & = & h_0(n,\vec{x},F(n,\vec{x})), \text{ if } n \neq 0 \\ F(2n+1,\vec{x}) & = & h_1(n,\vec{x},F(n,\vec{x})) \\ f(n,\vec{x}) & = & F(|n|,\vec{x}) \end{array}$$

provided that $F(n,\vec{x}) \leq k(n,\vec{x})$ holds for all $n, \vec{x}$.

The characterization of polynomial size, constant depth boolean circuits with parity gates (resp. MOD6 gates) uses sequence encoding techniques of A_0 together with logtime hierarchy analogues of work of Handley, Paris, Wilkie [42].

Theorem 24 (Clote-Takeuti [23])

$$\begin{array}{lrcl} (1) & \text{ACC}(2) & = & [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 1 - \text{BRN}] \\ (2) & \text{ACC}(6) & = & [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 2 - \text{BRN}] \\ (3) & \text{ACC}(6) & = & [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 3 - \text{BRN}]. \end{array}$$

The following characterization of $\mathcal{F}$ALOGTIME uses earlier techniques with a formalization of Barrington's trick [4] of expressing boolean connectives AND, OR by polynomial size bounded width branching programs.

Theorem 25 (P. Clote [17])

$$\mathcal{F}\text{ALOGTIME} = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 4 - \text{BRN}].$$

Theorem 26 (P. Clote [25])

$$\begin{array}{rcl} \text{NC} & = & [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{WBRN}] \\ \text{AC}^k & = & \{f \in \text{NC} : rk_{\text{WBRN}}(f) \leq k\}. \end{array}$$

It should be mentioned that independently and at about the same time, B. Allen [2] characterized NC by a function algebra using a form of *divide and conquer recursion*, and noticed without giving details that over a basis of appropriate initial functions, NC could also be characterized by the scheme of WBRN.[6] A precise statement of Allen's characterization is given later in Theorem 53.

Using such techniques, two characterizations of NC^k were given in [25, 23]. Levels of a natural time-space hierarchy between $\mathcal{F}$PTIME and $\mathcal{F}$PSPACE were characterized in [16].

4.3 Bounded recursion

In 1953, A. Grzegorczyk [39] investigated a hierarchy of subclasses $\mathcal{E}^n$ of primitive recursive functions, defined as the closure of certain initial functions under composition and bounded recursion.

Definition 27 The function f is defined by *bounded recursion* (BR) from functions g, h, k if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(x+1,\vec{y}) &= h(x,\vec{y},f(x,\vec{y})) \end{aligned}$$

provided that $f(x,\vec{y}) \le k(x,\vec{y})$ holds for all $x, \vec{y}$.

Definition 28 Let max be the binary maximum function. Define the following *principal* functions $f_0(x) = s(x) = x+1$, $f_1(x,y) = x+y$, $f_2(x,y) = (x+1)\cdot(y+1)$, $f_3(x) = 2^x$, and for $n \ge 3$ $f_{n+1}(x) = f_n^{(x)}(1)$, where the iterates of a function g are defined by $g^{(0)}(x) = x$ and $g^{(i+1)}(x) = g(g^{(i)}(x))$. Let $\mathcal{E}f$ denote $[0, I, s, f; \mathrm{COMP}, \mathrm{BR}]$ and $\mathcal{E}^{n+1}$ denote $\mathcal{E}f_{n+1}$.

In [39] Grzegorczyk proved that for all $n \ge 0$, $\mathcal{E}^n$ is properly contained in $\mathcal{E}^{n+1}$ by demonstrating that $f_{n+1} \notin \mathcal{E}^n$. Concerning the relational classes, he showed that for $n \ge 2$, $\mathcal{E}^n_*$ is properly contained in $\mathcal{E}^{n+1}_*$, and asked whether $\mathcal{E}^0_* \subset \mathcal{E}^1_* \subset \mathcal{E}^2_*$. This question remains open. In fact LTH $\subseteq \mathcal{E}^0_*$ and $\mathcal{E}^2_* =$ LINSPACE, so Grzegorczyk's question is related to the yet open problem whether the linear time hierarchy is properly contained in linear space. An interesting partial result concerning the containment of the first two relational classes is the following.

Theorem 29 (A. Bel'tyukov [8]) *For $s \ge 1$, let $\beta_s(x) = \max(1, x + \lceil x^{1-1/s} \rceil)$. Then for $s \ge 1$, $\mathcal{E}^0_* = (\mathcal{E}\beta_s)_*$. Additionally, $\mathcal{E}^2_* = \mathcal{E}^1_*$ implies $\mathcal{E}^2_* = \mathcal{E}^0_*$*

To obtain this result, Bel'tyukov introduced the *stack register machine*, a machine model capable of describing $(\mathcal{E}f)_*$. The stack register machine, a variant of the successor random access machine, has a finite number of *stack* registers $S_0, \ldots, S_k$ together with a *work* register W. Branching instructions

[6] See remark at bottom of p. 13 of [2].

```
if $p(x_1,\dots,x_m) = q(x_1,\dots,x_m)$ then $I_i$ else $I_j$
```

allow to jump to different instructions I_i, I_j depending on the comparison of two polynomials whose variables are current register values. Storage instructions

```
W = $S_i$
```

allow a value to be saved from a stack register to the work register. Incremental instructions

```
$S_i$ = $S_i$ + 1
```

perform the only computation, and have a side effect of setting to 0 all S_j for $j < i$. A program is a finite list of instructions, where for each i there is at most one incremental instruction for S_i.

Apart from characterizing $\mathcal{E}^2_*$ or LINSPACE, Bel'tyukov chacterized the linear time hierarchy LTH. The papers of Paris, Wilkie [71] and Handley, Paris, Wilkie [42] study counting classes between LTH and LINSPACE defined by stack register machines. Unpublished work of Handley [44, 43] and of the author [18] further study the effect of nondeterminism for this model.

The following characterization of LINSPACE in terms of the Grzegorczyk hierarchy was proved by R.W. Ritchie [73].

Theorem 30 $\mathcal{F}\text{LINSPACE} = \mathcal{E}^2$.

Theorem 31 (D.B. Thompson [86])

$$\mathcal{F}\text{PSPACE} = [0, I, s, \#; \text{COMP}, \text{BR}] = [0, I, s, \max, x^{|x|}; \text{COMP}, \text{BR}].$$

Definition 32 Let k be an integer. The function f is defined by *k-bounded recursion* (k-BR) from functions g, h, k if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(x+1,\vec{y}) &= h(x,\vec{y},f(x,\vec{y})) \end{aligned}$$

provided that $f(x,\vec{y}) \le k$ holds for all $x, \vec{y}$.

The following characterization results from the method of proof of Barrington's characterization of ALOGTIME [4, 3], arithmetization techniques of this paper, and work of Chandra, Kozen, Stockmeyer [14] implying that $\text{ATIME}(n^{O(1)}) = \text{PSPACE}$. In [13], J. Cai and M. Furst give a related characterization of PSPACE using *safe-storage* Turing machines, a model related to Bel'tyukov's earlier stack register machines.

Theorem 33 (P. Clote [18]) *For $k \ge 4$,*

$$\mathcal{F}\text{PSPACE} = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, k\text{-BR}].$$

In [89] K. Wagner extended Ritchie's characterization to more general complexity classes.

Theorem 34 (K. Wagner [89]) *Let f be an increasing function such that for some $r > 1$ and for all but finitely many x, it is the case that $f(x) \geq x^r$. Let $\mathcal{F}$ temporarily denote the algebra $[|f(2^n)|; \text{COMP}]$. Then*

$$\text{DSPACE}(\mathcal{F}) = [0, I, s, \max, f; \text{COMP}, \text{BR}]_* = [0, I, s, f_2, f; \text{COMP}, \text{BR}]_*.$$

The class DTIMESPACE$(n^{O(1)}, O(n))$ of simultaneous polynomial time and linear space can be characterized from the previous theorem by taking $f(x) = x^2$. As referenced in [90], S.V. Pakhomov [70] has characterized general complexity classes DTIMESPACE(T, S), DTIME(T), and DSPACE(S) for suitable classes S, T of unary functions.

The class QL $=$ DTIME$(n \cdot (\log n)^{O(1)})$ of *quasilinear time* was studied by C.P. Schnorr in [78]. In analogy, let *quasilinear space* be the class DSPACE$(n \cdot (\log n)^{O(1)})$. Though Theorem 34 characterizes quasilinear space via a function algebra, there appears to be no known function algebra for *quasilinear time.* In [41] Y. Gurevich and S. Shelah studied the class NLT (*nearly linear time*) of functions computable in time $O(n \cdot (\log n)^{O(1)})$ on a random access Turing machine RTM, which is allowed to change its input tape. Gurevich and Shelah show the robustness of NLT by proving the equivalence of this class with respect to different machine models, and give a function algebra for NLT. Their algebra, defined over words from a finite alphabet, is the closure under composition of certain initial functions and weak iterates $f^{(|x|)}(x)$ of certain string manipulating initial functions.

4.4 Bounded minimization

In [39], Grzegorczyk considered function classes defined by bounded minimization.

Definition 35 The function f is defined by bounded minimization (BMIN) from the function g, denoted $f(x, \vec{y}) = \mu i \leq x[g(i, \vec{y}) = 0]$, if

$$f(x, \vec{y}) = \begin{cases} \min\{i \leq x : g(i, \vec{y}) = 0\} & \text{if } (\exists i \leq x)(g(i, \vec{y}) = 0) \\ 0 & \text{else.} \end{cases}$$

For $n \geq 0$, define $\mathcal{M}^n = [0, I, s, f_n; \text{COMP}, \text{BMIN}]$.[7]

[7] In [75], following work of K. Harrow [46, 47], for $n \geq 3$, $\mathcal{M}^n$ is defined to be $[0, I, s, x^y, f_n; \text{COMP}, \text{BMIN}]$. By Theorem 41 and Proposition 38, the exponential x^y can be defined as $\mu z \leq g(x, y)(G(x, y, z) = 0)$, where g is obtained from f_n and initial functions by composition, and G is the characteristic function of the graph of exponentiation, which belongs to $\mathcal{M}^2_*$. For this reason, exponentiation is not included in our definition of $\mathcal{M}^n$.

The idea of proof of the following is simply to encode via sequence numbers a definition by bounded primitive recursion and apply the bounded minimization operator.

Theorem 36 (Grzegorczyk [39], Harrow [47]) *For $n \geq 3$, $\mathcal{E}^n = \mathcal{M}^n$.*

In the literature, the algebra RF of *rudimentary functions* is sometimes defined by

$$\text{RF} = [0, I, s, +, \times; \text{COMP}, \text{BMIN}].$$

As noticed in [47], it follows from J. Robinson's [74] bounded quantifier definition of addition from successor and multiplication that $\mathcal{M}^2 = \text{RF}$. As is well-known, there is a close relationship between (bounded) minimization and (bounded) quantification. Terms in the first order language of $0, s, +, \cdot, \leq$ of arithmetic are defined inductively by: 0 is a term; $x_0, x_1, \ldots$ are terms; if t, t' are terms, then $s(t)$, $t + t'$ and $t \cdot t'$ are terms. Atomic formulas are of the form $t = t'$ and $t \leq t'$, where t, t' are terms. The set Δ_0 of bounded quantifier formulas is defined inductively by: if ϕ is an atomic formula, then $\phi \in \Delta_0$; if $\phi, \theta \in \Delta_0$ then $\neg\phi$, $\phi \wedge \theta$, and $\phi \vee \theta$ belong to Δ_0; if $\phi \in \Delta_0$ and t is a term, then $(\exists x \leq t)\phi(x, t)$ and $(\forall x \leq t)\phi(x, t)$ belong to Δ_0.

Definition 37 A predicate $R \subseteq \mathbf{N}^k$ belongs to CA (*constructive arithmetic*), a notion due to R. Smullyan, if there is $\phi(\vec{x}) \in \Delta_0$ such that $R(a_1, \ldots, a_k)$ holds iff $\mathbf{N} \models \phi(a_1, \ldots, a_k)$. A function $f(\vec{x}) \in \mathcal{G}\text{CA}$ if the bitgraph $B_f \in \text{CA}$ and f is of linear growth.[8]

The following theorem is proved by using quantifier elimination for Presburger arithmetic to show the equivalence between first order formulas and bounded formulas in a richer language allowing congruences.

Theorem 38 (Harrow [46]) *$\mathcal{M}^1_*$ equals the collection of Presburger definable sets, $\mathcal{M}^2_* = \text{CA}$ and $\mathcal{M}^2 = \mathcal{G}\text{CA}$.*

Theorem 39 (J. Bennett [9]) *The graph $G_(x, y, z) \iff x^y = z$ of exponentiation is in constructive arithmetic.*

Corollary 40 *The function algebra $[0, I, s_0, s_1, |x|, \text{BIT}; \text{COMP}, \text{CRN}]$ is contained in $\mathcal{M}^2$.*

Theorem 41 (Bennett, Wrathall) LTH = CA.

Corollary 42 *$\mathcal{M}^2_* = \text{LTH}$, and $\mathcal{M}^2 = \mathcal{F}\text{LTH}$.*

[8] In the literature, especially in [71], a function f is defined to be Δ_0^N if its graph G_f belongs to Δ_0^N and f is of linear growth. It easily follows from Corollary 40 that $f \in \mathcal{G}\text{CA} \iff f \in \Delta_0^N$.

Though the linear time hierarchy equals the bounded arithmetic hierarchy, there is no known exact level-by-level result. The sharpest result we know is due to A. Woods [91].

If Γ is a class of first order formulas, then Γ^N denotes the collection of predicates definable by a formula in Γ. Let $\Sigma_{0,m}$ denote the collection of bounded quantifier formulas of the form $(\exists\vec{x}_1 \leq y)(\forall\vec{x}_2 \leq y)\dots(Q\vec{x}_m \leq y)\phi$ where ϕ is a quantifier free formula in the first order language $0, 1, +, \cdot, \leq$. Thus $\Sigma_{0,0}$ is the collection of quantifier free formulas.

Theorem 43 (A. Woods [91]) *For* $m \geq 1$ $\Sigma^N_{0,m} \subseteq \Sigma_{m+1} - \text{TIME}(O(n))$.

By Corollary 42 and Theorem 30, $\mathcal{M}^2_* = \text{LTH} \subseteq \text{LINSPACE} = \mathcal{E}^2_*$. While LINSPACE is clearly closed under *counting*, this may not be the case for LTH. A typical open question is whether $\pi(x) \in \mathcal{M}^2$, where $\pi(x)$ is the number of primes less than x. In [71, 42] J. Paris, A. Wilkie and later W. Handley studied the effect of adding k-bounded recursion to LTH. Using the techniques of Barrington, Paris, Wilkie and Handley, together with those of this paper, the following result can be proved.

Theorem 44 (P. Clote [18])
For any $k \geq 4$, $\text{ALINTIME}_{ra} = [0, I, s, +, \times; \text{COMP}, \text{BMIN}, k\text{-BR}]$.

As in Corollary 42, $\mathcal{F}\text{PH}$ can similarly be characterized.

Theorem 45 (Folklore)

$$\begin{aligned}\mathcal{F}\text{PH} &= [0, I, s, +, \times, \#; \text{COMP}, \text{BMIN}]\\ &= [0, I, s, +, \times, \#; \text{COMP}, \text{BRN}, \text{BMIN}].\end{aligned}$$

The second assertion of the last theorem was sharpened by S. Bellantoni as follows. Let Σ^P_i denote $\Sigma_i - \text{TIME}(n^{O(1)})$. Following S. Buss [11] let $\square^P_i$ denote the class of functions computed in polynomial time on a Turing machine with oracle A, for some set $A \in \Sigma^P_i$. With this notation, $\mathcal{F}\text{PH} = \cup_i \square^P_i$.

Theorem 46 (S. Bellantoni [6]) *For* $i \geq 0$,

$$\square^P_i = \{f \in [0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}, \text{BMIN}] : rk_{\text{BMIN}}(f) \leq i\}.$$

4.5 Divide and conquer, course-of-values and misc.

Definition 47 Let $\mathcal{R}_k$ be the smallest class of functions definable from the constant functions $0, \dots, k$, the projections I, the characteristic functions of the *graphs* of $+, \times, =$ and closed under composition and bounded recursion.

The following result was proved by the Paris-Wilkie modification of Bel'tyukov's stack register machines.

Theorem 48 (Paris, Wilkie [71]) $(\mathcal{R}_2)_* = (\mathcal{R}_3)_*$.

The next theorem follows from the author's work in [18] and is based on Barrington's trick.

Theorem 49 *For* $n \geq 4$,

$$(\mathcal{R}_n)_* = (\mathcal{R}_{n+1})_* = \text{ALINTIME}_{ra}.$$

In [61], Kutyłowski considered oracle versions of the Paris-Wilkie work.

Definition 50 (M. Kutyłowski [61]) f is a k-function[9] if for all $x_1, \ldots, x_n$

$$f(x_1, \cdots, x_n) = f(min(x_1, k), \cdots, min(x_n, k)) \leq k.$$

For a family $\mathcal{F}$ of functions, $\mathcal{W}_k(\mathcal{F})$ is the smallest class of functions containing I, $\mathcal{F}$, all k-functions and closed under composition and k-bounded recursion. The function f is defined from g, h by m-counting if

$$f(0, \vec{x}) = g(\vec{x})$$

$$f(n+1, \vec{x}) = (f(n, \vec{x}) + h(n, \vec{x})) \bmod m$$

The class $\mathcal{CW}_k(\mathcal{F})$ is the smallest class of functions containing I, $\mathcal{F}$, all m-functions for $m \in \mathbf{N}$ and closed under composition, k-bounded recursion and arbitrary counting.

Theorem 51 (M. Kutyłowski [61]) *For every class $\mathcal{F}$ of functions, $\mathcal{W}_2(\mathcal{F})_* = \mathcal{W}_3(\mathcal{F})_*$. For every $k \geq 3$, there exists a family $\mathcal{F}$ of functions, such that $\mathcal{W}_k(\mathcal{F})_* \subset \mathcal{W}_{k+1}(\mathcal{F})_*$. For every $k \geq 3$, there is a family $\mathcal{F}$ of functions such that $\mathcal{CW}_k(\mathcal{F})_* \subset \mathcal{CW}_{k+1}(\mathcal{F}_*)$.*

Parallel algorithms often employ a divide and conquer strategy. B. Allen [2] formalized this approach to characterize NC.

Definition 52 The *front half* FH(x) is defined by MSP$(x, \lfloor |x|/2 \rfloor)$ and the *back half* BH(x) by LSP$(x, \lfloor |x|/2 \rfloor)$. The function f is defined by *polynomially bounded branching recursion* (PBBR) from functions g, h if there exists a polynomial p such that

$$\begin{aligned} f(0, \vec{y}) &= g(0, \vec{y}) \\ f(1, \vec{y}) &= g(1, \vec{y}) \\ f(x, \vec{y}) &= h(x, \vec{y}, f(\text{FH}(x), \vec{y}), f(\text{BH}(x), \vec{y})), \text{if } y > 1 \end{aligned}$$

[9] What is here called a k-function is called a $k+1$-function in [61]. As our definition of k-bounded recursion corresponds to Kutyłowski's definition of $k+1$-bounded recursion, the indices of $\mathcal{W}_k(\mathcal{F})$ and $\mathcal{CW}_k(\mathcal{F})$ differ by 1 from [61].

provided that $|f(x,y_1,\ldots,y_m)| \leq p(\max(|x|,|y_1|,\ldots,|y_m|))$ for all $x,y_1,\ldots,y_m$. Let $Seq(x) = 0$ if x encodes a sequence[10] else 0. If x encodes a sequence $(x_1,\ldots,x_n)$ and f is a one-place function, then the operation MAP (similar to CRN) is defined by $\text{MAP}(f,x) = \langle f(x_1),\ldots,f(x_n)\rangle$. Define the *bounded shift left function* by $\text{SHL}(x,i,y) = x \cdot 2^{\min(i,|y|)}$.

Theorem 53 (B. Allen [2]) NC *is characterized by the function algebra*

$$[0, I, s, +, \dot{-}, |x|, \text{BIT}, cond, c_{\leq}, Seq, \beta, \text{MSP}, \text{SHL}; \text{COMP}, \text{MAP}, \text{PBBR}].$$

Allen explicitly did not attempt to find the smallest set of initial functions, but went on to develop a proof theory for NC functions, similar in spirit to that of S. Buss [11] (see also [22] for related work).

The Fibonacci sequence $1,1,2,3,5,8,\ldots$ is defined by $Fib(0) = Fib(1) = 1$, and $Fib(n+2) = Fib(n) + Fib(n+1)$. This is a special case of course-of-values recursion.

Definition 54 The function f is defined from functions g, h by *course-of-values recursion* (VR) if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(x+1,\vec{y}) &= h(x,\vec{y},\langle f(0,\vec{y}),\ldots,f(x,\vec{y})\rangle). \end{aligned}$$

The class $\mathcal{PR}$ of primitive recursive functions is easily seen to be closed under VR. For complexity classes, it is of more interest to consider a bounded version of course-of-values recursion, where $f(x+1)$ depends on at most two previously defined values of f.

Definition 55 The function f is defined from functions g, h, r, k by *bounded 2-value recursion* (BVR) if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(x+1,\vec{y}) &= h(x,\vec{y},f(x,\vec{y}),f(r(x,\vec{y}),\vec{y})) \end{aligned}$$

provided that $f(x,\vec{y}) \leq k(x,\vec{y})$ and $r(x,\vec{y}) < x$ for all $x, \vec{y}$.

Theorem 56 (Monien [66]) *Let* $f_2(x,y) = (x+1)\cdot(y+1)$, *and* ETIME *be* $\bigcup_{c\geq 1}\text{DTIME}(2^{c\cdot n})$. *Then*

$$\{f \in \text{ETIME} : f \text{ has linear growth rate}\} = [0, I, s, f_2; \text{COMP}, \text{BVR}].$$

When a pairing function is available, forms of simultaneous recursion can usually be deduced from corresponding forms of non-simultaneous recursion. A more powerful version of simultaneous recursion was introduced in [56].

[10] Allen [2] uses a different sequence encoding technique.

Definition 57 The functions $f_1, \ldots, f_n$ are defined from functions $g_1, \ldots, g_n$, $h_1^0, \ldots, h_n^0$, $h_1^1, \ldots, h_n^1$ and $k_1, \ldots, k_n$ by *multiple bounded recursion on notation* if the f_i are defined by simultaneous recursion on notation from $\vec{g_i}$, $\vec{h_i^0}$, $\vec{h_i^1}$ and moreover

$$\begin{array}{rcl} f_1(x, \vec{y}) & \leq & k_1(x, \vec{y}) \\ f_i(x, \vec{y}) & \leq & k_i(x, \vec{y}, f_1(x, \vec{y}), \ldots, f_{i-1}(x, \vec{y})), \text{ for } 2 \leq i \leq n. \end{array}$$

The following non-trivial closure property has an important application in the Kapron-Cook characterization of type 2 polynomial time computations described in the next section.

Theorem 58 (Kapron-Cook [56]) *The Cobham algebra* $[0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}]$ *is closed under multiple bounded recursion on notation.*

The following definition and theorem will be used in the next section to characterize the type 2 parallel complexity class NC.

Definition 59 The functions $f_1, \ldots, f_n$ are defined from $\vec{g}, \vec{h^0}, \vec{h^1}, \vec{k}$ by *multiple weak bounded recursion on notation* if $f_i(x, \vec{y}) = F_i(|x|, \vec{y})$, where $F_1, \ldots, F_n$ are defined by multiple bounded recursion on notation from $\vec{g}, \vec{h^0}, \vec{h^1}, \vec{k}$.

Recall that the algebra $A = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{WBRN}]$ coincides with the parallel complexity class NC, consisting of those functions computable in polylogarithmic time with a polynomial number of processors on a concurrent random access machine. The following was proved by the author and will appear in the journal version [19] of [20].

Theorem 60 *The algebra A is closed under multiple weak bounded recursion on notation.*

4.6 Safe recursion

All the function algebras from the previous subsection are defined from specific initial functions, using some version of bounded recursion. Without any bound, even schemes such as WBRN can generate all the primitive recursive functions. Recently, certain *unbounded* recursion schemes have been introduced which distinguish between variables as to their position in a function $f(x_1, \ldots, x_n; y_1, \ldots, y_m)$. Variables x_i occurring to the left of the semi-colon are called *normal*, while variables y_j to the right are called *safe*. By allowing only recursions of a certain form, which distinguish between normal and safe variables, particular complexity classes can be characterized. *Normal* values are considered as known in totality, while *safe* values are those obtained by impredicative means (i.e. via recursion). Sometimes, to help distinguish normal from safe positions, the letters $u, v, w, x, y, z, \ldots$ denote normal variables, while $a, b, c, \ldots$ denote safe variables. This terminology, due to Bellantoni-Cook [7], was chosen to indicate that a *safe* position is one where it is safe to substitute an impredicative value. Related *tiering* notions, though technically different, have occurred in the literature, as in [21] and most especially in work of D. Leivant (see [62]).

If $\mathcal{F}$ and $\mathcal{O}$ are collections of initial functions and operations which distinguish normal and safe variables, then NORMAL $\cap\ [\mathcal{F};\mathcal{O}]$ denotes the collection of all functions $f(\vec{x};) \in [\mathcal{F};\mathcal{O}]$ which have only normal variables. Similarly, (NORMAL $\cap\ [\mathcal{F};\mathcal{O}])_*$ denotes the collection of predicates whose characteristic function $f(\vec{x};)$ has only normal variables and belongs to $[\mathcal{F};\mathcal{O}]$.

Definition 61 (Bellantoni-Cook [7]) The function f is defined by *safe composition* (SCOMP) from $g, u_1, \ldots, u_n, v_1, \ldots, v_m$ if

$$f(\vec{x};\vec{a}) = g(u_1(\vec{x};), \ldots, u_n(\vec{x};); v_1(\vec{x};\vec{a}), \ldots, v_m(\vec{x};\vec{a})).$$

If $h(x;y)$ is defined, then SCOMP allows one to define

$$f(x,y;) = h(I_1^{2,0}(x,y;); I_2^{2,0}(x,y;)) = h(x;y).$$

However, one *cannot* similarly define $g(;x,y) = h(x;y)$.

Definition 62 The function f is defined by *safe recursion on notation*[11] (SRN) from the functions g, h_0, h_1 if

$$\begin{aligned} &f(0,\vec{y};\vec{a}) = g(\vec{y};\vec{a}) \\ &f(s_0(x),\vec{y};\vec{a}) = h_0(x,\vec{y};\vec{a}, f(x,\vec{y};\vec{a})), \text{ provided } x \neq 0 \\ &f(s_1(x),\vec{y};\vec{a}) = h_1(x,\vec{y};\vec{a}, f(x,\vec{y};\vec{a})). \end{aligned}$$

Define the following initial functions by

$$\begin{array}{rc} \text{(0-ary constant)} & 0 \\ \text{(projections)} & I_j^{n,m}(x_1,\ldots,x_n;a_1,\ldots,a_m) = \begin{cases} x_j & \text{if } 1 \leq j \leq n \\ a_{j-n} & \text{if } n < j \leq n+m \end{cases} \\ \text{(successors)} & S_0(;a) = 2 \cdot a,\ S_1(;a) = 2 \cdot a + 1 \\ \text{(binary predecessor)} & P(;a) = \lfloor a/2 \rfloor \\ \text{(conditional)} & C(;a,b,c) = \begin{cases} b & \text{if } a \bmod 2 = 0 \\ c & \text{else.} \end{cases} \end{array}$$

The function algebra B is defined by

$$[0, I, S_0, S_1, P, C; \text{SCOMP}, \text{SRN}].$$

[11] In [7] this scheme is called *predicative notational recursion.*

Theorem 63 (Bellantoni-Cook [7]) *The polynomial time computable functions are exactly those functions of B having only normal arguments, i.e.*

$$\mathcal{F}\text{PTIME} = \text{NORMAL} \cap B.$$

This approach has led to other characterizations of familiar complexity classes using *safe* variants of unbounded recursion schemes.

Theorem 64 (Bellantoni [5])

$$\{f \in \mathcal{F}\text{LOGSPACE} : |f(\vec{x})| = O(\log|x|)\} = \{f(\vec{x};) : f \in [0, I, S_1, P, C; \text{SCOMP}, \text{SRN}]\}.$$

Corollary 65

$$\text{LOGSPACE} = (\text{NORMAL} \cap [0, I, S_1, P, C; \text{SCOMP}, \text{SRN}])_*.$$

Definition 66 The function f is defined by *safe minimization* (SMIN) from the function g, denoted $f(\vec{x};\vec{b}) = \mu a[g(\vec{x};a,\vec{b}) \bmod 2 = 0)]$, if

$$f(\vec{x};\vec{b}) = \begin{cases} \min\{a : g(\vec{x};a,\vec{b}) = 0\}, & \text{if such exists,} \\ 0 & \text{else.} \end{cases}$$

The algebra $\mu B = [0, I, S_0, S_1, P, C; \text{SCOMP}, \text{SRN}, \text{SMIN}]$. Let μB_i denote the set of functions derivable in μB using at most i applications of safe minimization.

Theorem 67 (Bellantoni [6])

$$\Box_i^P = \{f(\vec{x};) : f \in \mu B_i\}.$$

Definition 68 (Bellantoni [5]) The function f is defined by *safe recursion*[12] (SR) from the functions g, h if

$$\begin{aligned} f(0,\vec{y};\vec{a}) &= g(\vec{y};\vec{a}) \\ f(x+1,\vec{y};\vec{a}) &= h(x,\vec{y};\vec{a},f(x,\vec{y};\vec{a})). \end{aligned}$$

Define the following initial functions by

$$\begin{array}{rl} \text{(successor)} & S(;a) = a+1 \\ \text{(predecessor)} & Pr(;a) = a \dot{-} 1 \\ \text{(conditional)} & K(;a,b,c) = \begin{cases} b & \text{if } a = 0 \\ c & \text{else.} \end{cases} \end{array}$$

Theorem 69 (Bellantoni [5])

$$\mathcal{E}^2 = \text{NORMAL} \cap [0, I, S, Pr, K; \text{SCOMP}, \text{SR}].$$

[12] In [5] this scheme is called *predicative primitive recursion.*

W. Handley (unpublished) independently obtained Theorem 69.

Turning to parallel computation, by building on Theorem 26, S. Bellantoni [5] characterizes NC as those functions with normal variables in an algebra built up from 0, I, S_0, S_1, the conditional C, the bit function BIT, the length function $L(;a) = |a|$, a variant $\#'$ of the smash function, and closed under safe composition, concatenation recursion on notation and a version of *safe* version of weak bounded recursion on notation. Define the *half* function by $H(x) = \lfloor x/(2^{\lceil |x|/2 \rceil}) \rfloor$, and note that the least number of times which H can be iterated on x before reaching 0 is $||x||$. The function f is defined by *safe weak recursion on notation*[13] from the functions g, h if

$$\begin{aligned} &f(0, \vec{y}; \vec{a}) = g(\vec{y}; \vec{a}) \\ &f(x, \vec{y}; \vec{a}) = h(x, \vec{y}; \vec{a}, f(H(x), \vec{y}; \vec{a})), \text{provided } x \neq 0. \end{aligned}$$

Theorem 70 (S. Bellantoni [5])

$$\text{NC} = [0, I, S_0, S_1, C, L, \text{BIT}, \#'; \text{SCOMP}, \text{CRN}, \text{SWRN}].$$

Following [2], define $\text{BH}(x) = x \bmod 2^{\lceil |x|/2 \rceil}$ and $\text{FH}(x) = msp(x, \text{BH}(x))$. The *back half* $\text{BH}(x)$ consists of the $\lceil |x|/2 \rceil$ rightmost bits of x, while the *front half* $\text{FH}(x)$ consists of the $\lfloor |x|/2 \rfloor$ leftmost bits of x. In [10] S. Bloch defines two distinct safe versions of Allen's divide and conquer recursion.

Definition 71 (S. Bloch [10]) The function f is defined by *safe divide and conquer recursion* (SDCR) from the functions g, h if The function f is defined by *very safe divide and conquer recursion* (VSDCR) from the functions g, h if

$$f(x, y, \vec{z}; \vec{a}) = \begin{cases} g(x, \vec{z}; \vec{a}) & \text{if } |x| \leq \max(|y|, 1) \\ h(; x, \vec{z}, \vec{a}, f(\text{FH}(; x), y, \vec{z}; \vec{a}), f(\text{BH}(; x), y, \vec{z}; \vec{a})) & \text{else.} \end{cases}$$

Note that in VSDCR the iteration function h has no normal parameters, and hence cannot itself be defined by recursion.

Theorem 72 (S. Bloch [10]) *There is a collection* BASE *of initial functions, for which*[14]

$$\text{ALOGTIME} = (\text{NORMAL} \cap [\text{BASE}; \text{SCOMP}, \text{VSDCR}])_*$$

$$\text{DSPACE}(\log^{O(1)} n) = (\text{NORMAL} \cap [\text{BASE}; \text{SCOMP}, \text{SDCR}])_*.$$

It seems clear that linear time on multitape Turing machines can be characterized using appropriate initial functions (sufficient to define NEXT_M) and closure under safe recursion and some form of very safe recursion. Details have been worked out by S. Bloch in unpublished work, and a related category theoretic characterization has been announced by J. Otto [69].

[13] In [5] this scheme is called *log recursion.*

[14] Bloch states his second result in terms of polylogarithmic parallel time (with no processor bound). Since parallel time equals sequential space, this is an equivalent assertion.

5 Type 2 functionals

Many programming languages allow functions to be passed as parameters to other functions or procedures. For instance, FORTRAN, PASCAL, and C allow function parameters, while C^{++} supports function templates and ADA, ML admit limited polymorphism.[15] The oracle Turing machine is a reasonable construct to model function parameter passing, though it has principally been used to study reducibilities $A \leq_T B$, $A \leq_T^P B$ etc. between *sets*. Nevertheless, higher type *functional* complexity theory is a new area with fundamental open problems. In particular, though various classes have been been proposed as candidates for the *feasible* type 2 functionals, there is not yet general agreement about the right notion. For reasons of space, only a few recent directions in higher type functional complexity will be presented. For more information, see the survey [28] by S.A. Cook.

Definition 73 A *type 2* functional F of *rank* (k, ℓ) is a total mapping from $(\mathbf{N}^N)^k \times \mathbf{N}^\ell$ into $\mathbf{N}$.

Definition 74 An oracle Turing machine (OTM) is a Turing machine M which in addition to read-only input tape, distinguished output tape and finitely many work tapes, has an *oracle query tape* and *oracle answer tape*, both one-way infinite, for each function input. Additionally M has a special oracle query state for each function input.

In order to query a function input f at x, the machine M takes steps to write x in binary on the oracle query tape. When the oracle query tape head is in its leftmost square, M enters a special query state. In the next step, M erases both the oracle query and answer tapes, writes the function value $f(x)$ in binary on the oracle answer tape, and leaves the oracle query and answer tape heads in their leftmost squares. Upon entering the oracle query state, there seem to be two natural measures for the time to complete the function query $f(x)$. The *unit cost*, considered by Mehlhorn [65], charges unit time, while the *function length cost*, considered by Constable [27] and later Kapron and Cook [56], charges $max\{1, |f(x)|\}$ time. The machine M computes the $rank(n, m)$ functional $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ if M has n oracle query states, query and answer tapes corresponding to $f_1, \ldots, f_n$ and if M outputs the integer $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ in binary on the output tape, when started in its initial state q_0 with input tape $\underline{B}x_1Bx_2B \cdots Bx_mB$.

Definition 75 An OTM M is a *polynomial time oracle Turing machine* (POTM) if M computes a total $rank(n, m)$ functional F and there is a polynomial p such that for all input $f_1, \ldots, f_n, x_1, \ldots, x_m$ and times t

$$t \leq p(|max(\{x_1, \ldots, x_m\} \cup A_M(\vec{f}, \vec{x}, t))|).$$

[15] Polymorphism allows function and procedures to abstract over data types — e.g. a generic sorting algorithm for any data type having a comparison function.

OPT is the collection of type 2 functionals computable by an oracle polynomial time oracle Turing machine.

Example 76

(1) $F(f,x) = max\{f(y) : y \leq |x|\}$ belongs to OPT.

(2) $G(f,x) = max\{f(y) : |y| \leq |x|\}$ does not belong to OPT.

(3) $H(f,x) = f^{(|x|)}(x)$ belongs to OPT.

In [65] K. Mehlhorn extended Cobham's function algebra to type 2 functionals. A modern presentation of Mehlhorn's definition uses the following schemes.

Definition 77 (Townsend [87]). F is defined from $H, G_1, \ldots, G_m$ by *functional composition* if for all $\vec{f}, \vec{x}$,

$$F(\vec{f},\vec{x}) = H(\vec{f}, G_1(\vec{f},\vec{x}), \ldots, G_m(\vec{f},\vec{x}), \vec{x}).$$

F is defined from G by *expansion* if for all $\vec{f}, \vec{g}, \vec{x}, \vec{y}$,

$$F(\vec{f},\vec{g},\vec{x},\vec{y}) = G(\vec{f},\vec{x}).$$

F is defined from G, H, K by *limited recursion on notation* (LRN) if for all $\vec{f}, \vec{x}, y$,

$$\begin{array}{rcl} F(\vec{f},\vec{x},0) & = & G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},y) & = & H(\vec{f},\vec{x},y,F(\vec{f},\vec{x},\lfloor\frac{y}{2}\rfloor)), \text{ if } y \neq 0 \end{array}$$

provided that $F(\vec{f},\vec{x},y) < K(\vec{f},\vec{x},y)$ holds for all $\vec{f},\vec{x},y$.

Definition 78 (Townsend [87], Kapron, Cook [29]) Let X be a class of type 2 functionals. The class of *basic feasible functionals* defined from X, denoted BFF(X), is the smallest class of functionals containing X, 0,s_0,s_1,i_k^n,$\#$ and the application functional Ap, defined by $Ap(f,x) = f(x)$, and which is closed under functional composition, expansion, and LRN. If $F \in$ BFF(X), then F is basic feasible in X. The class BFF of basic feasible functionals is BFF($\emptyset$).

Theorem 79 (Mehlhorn [65]) BFF *is the collection of type 2 functionals computable on an* OTM *with unit cost, where the runtime on input* $\vec{f},\vec{x}$ *is bounded by* $|F(\vec{f},\vec{x})|$ *for some* F *belonging to* BFF.

It is clear that OPT contains functionals which are not intuitively feasible. In particular, substituting the polynomial time computable function $\lambda y.y^2$ for f in H, where $H(f,x) = f^{(|x|)}(x)$, above yields $H(\lambda y.y^2, x) = x^{2^{|x|}}$ which is not a polytime computable type 1 function (example due to A. Seth [81]). The following example, due to S. Cook, provides a functional which belongs to OPT yet not to BFF.

Let $\preceq$ quasi-order $\mathbf{N} \times \mathbf{N}$ by *length first difference*; i.e. $(a,b) \preceq (c,d)$ iff $|a| < |c|$ or ($|a| = |c|$ and $|b| \leq |d|$). Transfer this ordering to $\mathbf{N}$ by a standard polynomial time pairing function. Define the $rank(1,0)$ functional L by $L(f) = \mu i[(\exists j < i)(f(j) \preceq f(i))]$. Note that $\preceq$ defines a quasi-well ordering on $\mathbf{N} \times \mathbf{N}$, so L is well defined.

Theorem 80 (S. Cook [28]) *The functional L belongs to* OPT *yet not to* BFF.

In [56], Kapron and Cook lift Cobham's characterization of polynomial time computable functions to functionals of level 2. To state their result, the notion of *length* of a function and that of second order polynomial must be introduced.

Definition 81 The length $|f|$ of one-place function f is itself a one-place function defined by

$$|f|(n) = \max_{|x| \leq n}\{|f(x)|\}.$$

Let $f_1, \ldots, f_m$ be variable ranging over $\mathbf{N}^N$ and $x_1, \ldots, x_n$ be variables ranging over $\mathbf{N}$. The collection C of second order polynomials $P(f_1, \ldots, f_m, x_1, \ldots, x_n)$ is defined inductively as follows.

(i) for any integer c, $c \in C$,
(ii) for every $1 \leq i \leq n$, $x_i \in C$,
(iii) if $P, Q \in C$ then $P + Q \in C$ and $P \cdot Q \in C$,
(iv) if $P \in C$ then $f_i(P) \in C$ for $1 \leq i \leq m$.

Theorem 82 (B. Kapron and S. Cook [56]) BFF *is the collection of functionals $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ computable in time $P(|f_1|, \ldots, |f_n|, |x_1|, \ldots, |x_m|)$ for some second order polynomial P on an* OTM *with function length cost.*[16]

The *oracle concurrent random access machine* (OCRAM), introduced in [20] has instructions for *(i)* local operations — addition, cutoff subtraction, shift, *(ii)* global and local indirect reading and writing, *(iii)* control instructions — GOTO, conditional GOTO and HALT, *(iv)* oracle calls, where in one step, all active processors simultaneously can retrieve

$$f(x_i \cdots x_j) = f(\sum_{k=i}^{j} x_k \cdot 2^{j-k})$$

where i, j are current values of local registers, and x_i is the 0,1 value held in the i-th oracle register. The formal details of the this model ensure that the size of

[16] In [51], A. Ignjatovic has given an alternate proof theoretic proof of Theorem 82 and moreover has shown the same result to hold for unit cost.

the function value returned in any oracle call will be bounded by the product of the number of active processors and the total computation time.

The OCRAM is formally defined as follows. For each k-ary function argument f, there are k infinite collections of *oracle registers*, the i-th collection labeled $M_0^{o,i}, M_1^{o,i}, M_2^{o,i}, \ldots$, for $1 \leq i \leq k$. As with global memory, in the event of a write conflict the lowest numbered processor succeeds in writing to an oracle register. Let *res* (result), *op*0 (operand 0) and *op*1 (operand 1) be non-negative integers, as well as *op*2,*op*3,...,*op*(2*k*).

In addition to the instructions for the CRAM, the OCRAM has instructions concerning the oracle registers and oracle calls.

$$\begin{aligned}
*M_{res}^o &:= 0 \\
*M_{res}^o &:= 1 \\
M_{res}^o &:= 0 \\
M_{res}^o &:= 1 \\
M_{res} &:= *M_{op1}^o \\
M_{res} &:= f([M_{op1} \cdots M_{op2}]_1, [M_{op3} \cdots M_{op4}]_2, \ldots, [M_{op(2k-1)} \cdots M_{op(2k)}]_k)
\end{aligned}$$

The notation $[M_{op(2i-1)} \cdots M_{op(2i)}]_i$ denotes the integer whose binary notation is given in oracle registers $M_{M_{op(2i-1)}}^{o,i}$ through $M_{M_{op(2i)}}^{o,i}$.

In characterizing AC^k in the non-oracle case, Stockmeyer and Vishkin [85] require a polynomial bound $p(n)$ on the number of active processors on inputs of length n. With the above definition of OCRAM one might hope to characterize the class of type 2 functionals computable in constant parallel time with a second-order polynomial number of processors as exactly the type 2 functionals in the algebra $\mathcal{A}_0$. Using the definitions given so far, this is not true. To rectify this situation, proceed as follows.

Definition 83 For every OCRAM M, functions f, g and integers x, t the query set $Q(M, f, x, t, g)$ is defined as

$$\begin{aligned}
\{y : \ & M \text{ with inputs } f, x \text{ queries } f \text{ at } y \text{ in } < t \text{ steps, where for each} \\
& i < t \text{ the active processors are those with index } 0, \ldots, g(i) - 1\}.
\end{aligned}$$

Let M be an OCRAM, P a functional of rank (1,1), f a function and x, t integers. If $Q \subseteq \mathbf{N}$ then define $f_Q(u) = f(u)$ if $u \in Q$, otherwise 0. Define $\mathcal{M} = \langle M, P \rangle$ to be a *fully specified* OCRAM if for all f, x, t the OCRAM M on input f, x either is halted at step t or executes at step t with active processors $0, \ldots, P(|f_{Q_t}|, |x|) - 1$ where

$$Q_t = Q(M, f, x, t, P(|f_{Q_{t-1}}|, |x|))$$

is the collection of queries made by $\mathcal{M}$ before step t.

If $\mathcal{M} = \langle M, P \rangle$ is a fully specified OCRAM with input f, x define

$$Q_{\mathcal{M}}(f,x,t) = \{y : \mathcal{M} \text{ queries } y \text{ at time } i < t \text{ on input } f, x\}.$$

In place of stating that $\mathcal{M} = \langle M, P \rangle$ is fully specified, usually M is said to run with processor bound P. If $F(\vec{f}, \vec{x})$ abbreviates $F(f_1, ..., f_m, x_1, ..., x_n)$ and P is a second order polynomial, then $P(|\vec{f}|, |\vec{x}|)$ abbreviates $P(|f_1|, ..., |f_m|, |x_1|, ..., |x_n|)$.

The type 2 analogue of concatenation recursion on notation is given by the following.

Definition 84 F is defined from G, H, K by concatenation recursion on notation (CRN) if for all $\vec{f}, \vec{x}, y$,

$$\begin{array}{rcl} F(\vec{f},\vec{x},0) & = & G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},s_0(y)) & = & F(\vec{f},\vec{x},y)^\frown\text{BIT}(0,H(\vec{f},\vec{x},y)), \text{ provided that } x \neq 0 \\ F(\vec{f},\vec{x},s_1(y)) & = & F(\vec{f},\vec{x},y)^\frown\text{BIT}(0,K(\vec{f},\vec{x},y)). \end{array}$$

Definition 85 The type 2 functional H is defined by weak bounded recursion on notation WBRN from G, H_0, H_1, K if

$$\begin{array}{rcl} F(\vec{f},\vec{x},0) & = & G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},s_0(y)) & = & H_0(\vec{f},\vec{x},y,F(\vec{f},\vec{x},y)), \text{ if } n \neq 0 \\ F(\vec{f},\vec{x},s_1(y)) & = & H_1(\vec{f},\vec{x},y,F(\vec{f},\vec{x},y)) \\ H(\vec{f},\vec{x},y) & = & F(\vec{f},\vec{x},|y|) \end{array}$$

provided that $F(\vec{f},\vec{x},y) \leq K(\vec{f},\vec{x},y)$ holds for all $\vec{f},\vec{x},y$.

Definition 86 The algebra $\mathcal{A}_0$ is the smallest class of functionals (of type 1 and 2) containing 0,s_0,s_1,i^n_k,BIT,$|x|$,#, Ap and closed under functional composition, expansion, and CRN. The algebra $\mathcal{A}$ is the closure of 0,s_0,s_1,i^n_k,BIT,$|x|$,#, Ap under functional composition, expansion, CRN and WBRN.

The following theorem is the type 2 analogue of the fact that AC^0 (or equivalently LH) is characterized by the function algebra $\mathcal{A}_0$.

Theorem 87 (Clote, Kapron, Ignjatovic [20]) *A functional $F(\vec{f},\vec{x})$ belongs to $\mathcal{A}_0$ if and only if it is computable on an* OCRAM *in constant time with at most $P(|\vec{f}|,|\vec{x}|)$ many processors, for some second-order polynomial P.*

The type 2 analogue of Theorem 26 was established by the author and will appear in the journal version of [20]. For notational brevity, the theorem is stated for rank $(1,1)$ functionals, though an appropriate statement holds for type 2 functionals of any rank.

Theorem 88 (P. Clote) *Let F be a rank $(1,1)$ functional. Then $F \in \mathcal{A}$ if and only if there exist second order polynomials P, Q for which F is computed by an* OCRAM *running in polylogarithmic time $O(|P(|f|,|x|)|^k)$ with a polynomial $Q(|f|,|x|)$ number of processors.*

In his attempted proof of the continuum hypothesis, D. Hilbert [48] studied classes of higher type functionals defined by the operations of composition and primitive recursion. Hilbert's general scheme ([48], p. 186) was of the form

$$\mathcal{F}(G,H,0) = H$$
$$\mathcal{F}(G,H,n+1) = G(\mathcal{F}(G,H,n),n)$$

where $\mathcal{F}, G, H$ are higher type functionals of appropriate types possibly having other parameters not indicated. Illustrating the power of primitive recursion over higher type objects, Hilbert gave a simple higher type primitive recursive definition of the Ackermann function.

Higher type functional complexity theory is an emerging field. For reasons of space, only references to a few recent papers will be given. In [59] Ker-I Ko gave a survey of results concerning sequential complexity theory of real valued functions. In [49, 50], H.J. Hoover investigated parallel computable real valued functions. In [28], S. Cook gave a survey of higher type computational approaches, and proves Theorem 80. Cook further proposed that any class C of feasible type 2 functionals must satisfy the following two conditions: (1) BFF $\subseteq C \subseteq$ OPT, (2) C is closed under abstraction and application. In [81] A. Seth defined a class C_2 of type 2 functionals defined by *counter* Turing machines with polynomial bounds, which satisfies the previous conditions, and proved that no recursively presentable class of functionals exists which contains C_2 and satisfies the previous conditions. In [82] Seth further investigated closure conditions for feasible functionals. In [76], J. Royer studied a polynomial time counterpart to the Kreisel-Lacombe-Shoenfield theorem [60].

Complexity theory for functionals of all finite types was initiated by S. Buss, who in [12] introduced a polynomial time analogue of the *hereditarily recursive operations* HRO to define polynomial time functionals of all finite types decorated with runtime bounds. A. Nerode, J. Remmel and A. Scedrov [67] studied a polynomially graded type system. In [34], J.-Y. Girard, A. Scedrov and P. Scott introduced *bounded linear logic*, and prove a normalization theorem which yields a characterization of a feasible class of type 2 functionals. In [30] S. Cook and A. Urquhart introduced an analogue of Gödel's system **T** by admitting a recursor for bounded recursion on notation for type 1 objects. Their system PV^ω provided a natural class of polynomial time higher type functionals (called the *basic feasible functionals of higher type*). In [45], V. Harnik extended Cook-Urquhart's functionals to levels of the polynomial time hierarchy. In [29] S. Cook and B. Kapron characterized the higher type functionals in PV^ω by certain kinds of programming language constructs, *typed while* programs and *bounded loop* programs. This kind of characterization was extended by P. Clote, B. Kapron and A. Ignjatovic in [20] to the higher type functionals in NC^ω, relating *bounded loop* programs with higher type parallel complexity classes. In [83] A. Seth extended his definition of *counter* Turing machine to all finite types, thus characterizing PV^ω by a machine model. If one additionally allows dynamic computation of indices of subprograms within this counter Turing machine model, then Seth has conjectured this class to properly contain PV^ω.

In [63] D. Leivant and J.-Y. Marion gave various characterizations of PTIME by typed λ-calculi with pairing over an algebra $\mathbf{W}$ of words over $\{0,1\}$. In an unpublished paper, the same authors showed how a natural restriction of functional recurrence with substitution generates exactly PSPACE. In a series of papers (see for instance [62]) D. Leivant investigated various tiering schemes of recursion (extensions of safe recursion) and related complexity classes. Such investigations may have some applicability to programming language design. In [68], building on work of H. Schwichtenberg [80], K.-H. Niggl investigated certain subrecursive hierarchies (analogues of primitive recursive) of partial continuous functionals on Scott domains. Higher type functional complexity is currently an active field and likely to remain so for some time.

References

[1] W. Ackermann. Zum Hilbertschen Aufbau der reelen Zahlen. *Mathematische Annalen*, 99:118–133, 1928.

[2] B. Allen. Arithmetizing uniform NC. *Annals of Pure and Applied Logic*, 53(1):1–50, 1991.

[3] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity in NC^1. *Journal of Computer and System Science*, 41(3):274–306, 1990.

[4] D.A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[5] S. Bellantoni. Predicative recursion and computational complexity. Technical Report 264/92, University of Toronto, Computer Science Department, September 1992. 164 pages.

[6] S. Bellantoni. Predicative recursion and the polytime hierarchy. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 15–29. Birkhäuser, 1995.

[7] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[8] A. Bel'tyukov. A computer description and a hierarchy of initial Grzegorczyk classes. *Journal of Soviet Mathematics*, 20:2280 – 2289, 1982. Translation from Zap. Nauk. Sem. Lening. Otd. Mat. Inst., V. A. Steklova AN SSSR, Vol. 88, pp. 30 - 46, 1979.

[9] J.H. Bennett. *On Spectra.* PhD thesis, Princeton University, 1962. Department of Mathematics.

[10] S. Bloch. Function-algebraic characterizations of log and polylog parallel time. *Computational Complexity*, 4(2):175–205, 1994.

[11] S. Buss. *Bounded Arithmetic*, volume 3 of *Studies in Proof Theory*. Bibliopolis, 1986. 221 pages.

[12] S. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In A.L. Selman, editor, *Structure in Complexity Theory*, volume 223, pages 77–103. 1986. Springer Lecture Notes in Computer Science.

[13] J.-Y. Cai and M.L. Furst. $PSPACE$ survives three-bit bottlenecks. In *Proceedings of 3th Annual IEEE Conference on Structure in Complexity Theory*, pages 94–102, 1988.

[14] A. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association of Computing Machinery*, 28:114 – 133, 1981.

[15] A. Church. An unsolvable problem in elementary number theory. *American Journal of Mathematics*, 58:345 – 363, 1936.

[16] P. Clote. A time-space hierarchy between P and PSPACE. *Mathematical Systems Theory*, 25:77–92, 1992.

[17] P. Clote. Polynomial size frege proofs of certain combinatorial principles. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 162 – 184. Oxford University Press, 1993.

[18] P. Clote. Nondeterministic stack register machines. Submitted.

[19] P. Clote, B. Kapron, and A. Ignjatovic. Parallel computable higher type functionals. Technical Report BCCS-94-04, Department of Computer Science, Boston College, June 1994.

[20] P. Clote, B. Kapron, and A. Ignjatovic. Parallel computable higher type functionals. In *Proceedings of IEEE 34th Annual Symposium on Foundations of Computer Science*, Nov 3–5, 1993. Palo Alto CA. pp. 72–83.

[21] P. Clote and G. Takeuti. Exponential time and bounded arithmetic. In A.L. Selman, editor, *Structure in Complexity Theory*, volume 223, pages 125–143. Springer Lecture Notes in Computer Science, 1986.

[22] P. Clote and G. Takeuti. Bounded arithmetics for NC, $ALOGTIME$, L and NL. *Annals of Pure and Applied Logic*, 56:73–117, 1992.

[23] P. Clote and G. Takeuti. First order bounded arithmetic and small boolean circuit complexity classes. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 154–218. Birkhäuser Boston Inc., 1995.

[24] P.G. Clote. A sequential characterization of the parallel complexity class NC. Technical Report BCCS-88-07, Department of Computer Science, Boston College, 1988.

[25] P.G. Clote. Sequential, machine-independent characterizations of the parallel complexity classes $ALOGTIME, AC^k, NC^k$ and NC. In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 49–70. Birkhäuser, 1990.

[26] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science II*, pages 24–30. North-Holland, 1965.

[27] R. Constable. Type 2 computational complexity. In *5th Annual ACM Symposium on Theory of Computing*, 1973. pp. 108–121.

[28] S. Cook. Computability and complexity of higher type functions. In Y.N. Moschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer Verlag, 1992.

[29] S.A. Cook and B.M. Kapron. Characterizations of the feasible functionals of finite type. In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 71–98. Birkhäuser, 1990.

[30] S.A. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63(2):pp. 103–200, 1993.

[31] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[32] R. Fagin. Finite-model theory—a personal perspective. In S. Abiteboul and P. Kanellakis, editors, *Proc. 1990 International Conference on Database Theory*, pages 3–24. Springer-Verlag Lecture Notes in Computer Science 470, 1990. Journal version to appear in *Theoretical Computer Science*.

[33] S. Fortune and J. Wyllie. Parallelism in random access machines. In *10th Annual ACM Symposium on Theory of Computing*, 1978. pp. 114-118.

[34] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic. In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 195–210. Birkhäuser, 1990.

[35] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *J. Monat. Math. Phys.*, 38:173 – 198, 1931.

[36] K. Gödel. Conversation with G.E. Sacks. Institute for Advanced Study, 1975.

[37] L. Goldschlager. Synchronous parallel computation. Technical Report 114, University of Toronto, December 1977. 131 pages.

[38] L. Goldschlager. A unified approach to models of synchronous parallel machines. *Journal of the Association of Computing Machinery*, 29(4):pp. 1073–1086, October 1982.

[39] A. Grzegorczyk. Some clases of recursive functions. *Rozprawy Matematyczne*, 4, 1953.

[40] Y. Gurevich. Algebras of feasible functions. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*, 1983. pp. 210-214.

[41] Y. Gurevich and S. Shelah. Nearly linear time. *Symposium on Logical Foundations of Computer Science*, Springer Lecture Notes in Computer Science(363):108–118, 1989. Pereslavl-Zalessky, USSR.

[42] W. Handley, J. B. Paris, and A. J. Wilkie. Characterizing some low arithmetic classes. In *Theory of Algorithms*, pages 353 – 364. Akademie Kyado, Budapest, 1984. Colloquia Societatis Janos Bolyai.

[43] W.G. Handley. LTH plus nondeterministic summation mod M_3 yields ALINTIME. Submitted, 22 December 1994.

[44] W.G. Handley. Deterministic summation modulo $\mathcal{B}_n$, the semi-group of binary relations on $\{0, 1, \ldots, n-1\}$. Submitted, May 1994.

[45] V. Harnik. Provably total functions of intuitionistic bounded arithmetic. *Journal of Symbolic Logic*, 57(2):466–477, 1992.

[46] K. Harrow. Small Grzegorczyk classes and limited minimum. *Zeit. Math. Logik*, 21:417–426, 1975.

[47] K. Harrow. Equivalence of some hierarchies of primitive recursive functions. *Zeit. Math. Logik*, 25:411–418, 1979.

[48] D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1925.

[49] H. James Hoover. Feasibly constructive analysis. Technical Report 206/87, University of Toronto, November 1987. 114 pages.

[50] H.J. Hoover. Computational models for feasible real analysis. In S.R. Buss and P.J. Scott, editors, *Feasible Mathematics*, pages 221–238. Birkhäuser, 1990.

[51] A. Ignjatovic. Some applications of logic to feasibility in higher types. Typescript and invited talk at meeting LCC, Indianapolis, organizer D. Leivant, October 13–16 1994.

[52] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[53] N. Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989.

[54] N.D. Jones and A.L. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.

[55] L. Kalmar. Egyszerü példa eldönthetetlen aritmetikai problémára. *Mate és Fizikai Lapok*, 50:1–23, 1943. [In Hungarian with German abstract].

[56] B. Kapron and S. Cook. A new characterization of Mehlhorn's poly time functionals. In *Proceedings of IEEE 32th Annual Symposium on Foundations of Computer Science*, pages pp. 342–347, 1991. to appear in *SIAM J. on Comput.*

[57] S.C. Kleene. General recursive functions of natural numbers. *Math. Ann.*, 112:727–742, 1936.

[58] S.C. Kleene. Lambda-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[59] Ker-I Ko. Applying techniques of discrete complexity theory to numerical computation. In R.V. Book, editor, *Studies in Complexity Theory*, pages 1–62. John Wiley and Sons, Inc, 1986.

[60] G. Kreisel, D. Lacombe, and J.R. Shoenfield. Partial recursive functionals and effective operations. In A. Heyting, editor, *Constructivity in Mathematics: Proceedings of a colloquium held in Amsterdam*, pages 195–207. North Holland, 1957.

[61] M. Kutyłowski. Finite automata, real time processes and counting problems in bounded arithmetics. *Journal of Symbolic Logic*, 53(1):243–258, 1988.

[62] D. Leivant. Ramified recurrence and computational complexity I: word recurrence and poly-time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

[63] D. Leivant and J.-Y. Marion. Lambda-calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167–184, 1993.

[64] J.C. Lind. Computing in logarithmic space. Technical Report Project MAC Technical Memorandum 52, Massachusetts Institute of Technology, September 1974.

[65] K. Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Science*, 12:147–178, 1976.

[66] B. Monien. A recursive and grammatical characterization of exponential time languages. *Theoretical Computer Science*, 3:61–74, 1977.

[67] A. Nerode, J. Remmel, and A. Scedrov. Polynomially graded logic I – a graded version of system T. In *Proceedings of IEEE 4th Annual Symposium on Logic in Computer Science*, 1989.

[68] K.-H. Niggl. Subrecursive hierarchies on Scott domains. *Archive for Mathematical Logic*, 32:239–257, 1993.

[69] J. Otto. Tiers, tensors, and Δ_0^0. Talk at meeting LCC, Indianapolis, organizer D. Leivant, October 13–16 1994.

[70] S.V. Pakhomov. Machine independent description of some machine complexity classes (in Russian). *Issledovanija po konstrukt. matemat. i mat. logike*, VIII:176–185, LOMI 1979.

[71] J. B. Paris and A. J. Wilkie. Counting problems in bounded arithmetic. In C. A. di Prisco, editor, *Methods in Mathematical Logic*, pages 317 – 340. Springer Verlag Lecture Notes in Mathematics, 1983. Proceedings of Logic Conference held in Caracas, 1983.

[72] R. Péter. Über die mehrfache Rekursion. *Mathematische Annalen*, 113:489–526, 1936.

[73] R.W. Ritchie. Classes of predictably computable functions. *Trans. Am. Math. Soc.*, 106:139–173, 1963.

[74] J. Robinson. Definability and decision problems in arithmetic. *Journal of Symbolic Logic*, 14:98–114, 1949.

[75] H. E. Rose. *Subrecursion: Function and Hierarchies*, volume 9 of *Oxford Logic Guides*. Clarendon Press, Oxford, 1984. 191 pages.

[76] J.S. Royer. Semantics vs. syntax vs. computation. Typescript, November 29, 1994.

[77] W.L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22:pp. 365–383, 1981.

[78] C. P. Schnorr. Satisfiability is quasilinear complete in *NQL*. *Journal of the Association of Computing Machinery*, 25(1):136–145, 1978.

[79] H. Scholz. Ein ungelöstes Problem in der symbolischen Logik. *Journal of Symbolic Logic*, 17:160, 1952.

[80] H. Schwichtenberg. Primitive recursion on the partial continuous functionals. In M. Broy, editor, *Informatik und Mathematik*, pages 251–259. Springer-Verlag, 1991.

[81] A. Seth. There is no recursive axiomatization for feasible functionals of type 2. In *Proceedings of IEEE 7th Annual Symposium on Logic in Computer Science*, 1992. pp. 286–295.

[82] A. Seth. Some desirable conditions for feasible functionals of type 2. In *Proceedings of IEEE 8th Annual Symposium on Logic in Computer Science*, 1993.

[83] A. Seth. Turing machine characterizations of feasible functionals of all finite types. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 407–428. Birkhäuser, 1994.

[84] Y. Shiloach and U. Vishkin. Finding the maximum, merging and sorting in a parallel computation model. *Journal of Algorithms*, 3:57–67, 1982.

[85] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13:409–422, 1984.

[86] D.B. Thompson. Subrecursiveness: machine independent notions of computability in restricted time and storage. *Math. Systems Theory*, 6:3–15, 1972.

[87] M. Townsend. Complexity for type-2 relations. *Notre Dame Journal of Formal Logic*, 31:241–262, 1990.

[88] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc., Series 2*, 42:230–265, 1936-37.

[89] K. Wagner. Bounded recursion and complexity classes. In *Lecture Notes in Computer Science*, volume 74, pages 492–498. Springer-Verlag, 1979.

[90] K. Wagner and G. Wechsung. *Computational Complexity*. Reidel Publishing Co., 1986.

[91] A. Woods. Bounded arithmetic formulas and Turing machines of constant alternation. In J.B. Paris, A.J. Wilkie, and G.M. Wilmers, editors, *Logic Coloquium 1984*. North Holland, 1986.

[92] C. Wrathall. Complete sets and the polynomial time hierarchy. *Theoretical Computer Science*, 3:23 – 33, 1976.

Expressing Computational Complexity in Constructive Type Theory*

Robert L. Constable
Cornell University

Abstract

It is notoriously hard to express computational complexity properties of programs in programming logics based on a semantics which respects extensional function equality. This is a serious impediment to certain key applications of programming logics, even those which apply very well otherwise.

This paper shows how to define computational complexity measures in such logics as long as they support inductively defined types, dependent products, and functions. The method exploits a natural feature of inductive definitions in type theory, namely that implicit codes are kept with the objects showing how they are presented in the inductive class.

The adequacy of the proposed definition depends on a *faithfulness theorem* showing that the external (or meta-level) definition of complexity is respected by the internal definition. The results are applied to defining resource bounded quantifiers that can be used to state complexity constraints on constructive proofs and their extracted programs. In such *resource bounded logics* it is possible to prove theorems like a PTime axiom of choice. The results of the paper bridge the fields of semantics and complexity to a small extent.

1 Introduction

Most programming logics use this rule for function equality

$$f =_{A \to B} g \text{ iff } \forall x : A.\ f(x) =_B g(x).$$

The functions f and g may be given by programs, say that f and g are also names for the programs. In the meta-theory of the programming logic, we have access to a finer equality, the equality on f and g as programs or terms. Given

*Work supported in part by NSF grant CCR-9244739 and ONR grant N00014-92-J-1764

this access to the program structure, we can define the usual computational complexity measures, say *time*(f) and *space*(f). But in the object theory, we lose access to these functions since they do not respect the function equality.

One approach to gain access to computational complexity in the object logic is to define a finer equality on functions, say some intensional equality [10]. But experience has shown that such logics are difficult to use and to interface with conventional mathematics. Another approach is to use a logic, say Bounded Linear Logic [16], that keeps track of computational resources. This approach also requires a great deal of as yet unfinished work to show that such an axiomatization of programming is manageable.

The issue in this paper is to look for an *existing mechanism* in a constructive programming logic that can be exploited to define computational complexity in the object logic in a natural way.

The basic idea is to notice that the computational interpretation of an inductively defined class of functions, say $\mathcal{C}(A \rightarrow B)$ defined over $A \rightarrow B$, contains in it an implicit system of codes that can be used to define complexity. We will show that given $f \in \mathcal{C}(A \rightarrow B)$ we can define *time*(f) and *space*(f) in terms of these codes. This is possible because equality on inductive classes is not the same as equality in the underlying type. We are exploiting a fact that is naturally part of constructive inductive definitions, not introducing a new mechanism just for the sake of defining complexity. This mechanism is also present in programming languages like ML that support inductive types.

2 Basic Concepts

The results are given for an especially simple but powerful type theory. Its base types are just *unit*, **1**, and *void* (true and false under propositions-as-types) and the types $Type_i$. The element of *unit* is just a dot, $\cdot$. The constructors are *dependent* functions (Π) and dependent products (Σ). We denote the language by $\Pi\mu\Sigma^+$ (a Greek acronym for "Promise").

$\Pi x : A.\ B(x)$ are those functions $\lambda(x.\ b)$ such that for all $a \in A$, $f(a) \in B(a)$ ($f(a)$ is also written $ap(f,\ a)$).

$\Sigma x : A.\ B(x)$ are those pairs $< a,\ b >$ such that $a \in A$ and $b \in B(a)$.

It must be that A is a type and $B(a)$ is a type for each $a \in A$. We also allow the disjoint union, $A + B$, of types A and B. The elements are the *injections*, *inl*(a) and *inr*(b).

Finally, $\mu(X.\ F)$ is a recursive type, in $Type_i$, provided F is a monotone function $Type_i$ to $Type_i$. We say that $a \in \mu(X.\ F)$ iff $a \in F[\mu(X.\ F)/X]$. With each recursive type $\mu(X.\ F)$ there is a recursion combinator, μ–*ind*, which defines the $\mu(X.\ F)$ –recursive functions. The typing rule is

$$\frac{H \vdash a \in \mu(X.\ F) \quad X : Type_i,\ z : F,\ f : X \rightarrow G \quad \vdash g \in G}{H \vdash \mu\text{–}ind(a; z,\ f.\ g) \in G}$$

examples

example 1: The natural numbers can be defined as $\mu(N.\ \mathbf{1}+N)$ with $0 = inl(\cdot)$ and $succ(x) = inr(x)$. The primitive recursive functions are defined by the combinator μ–*ind* (details are not critical to the results).

example 2: We define the set of functions $N \rightarrow N$ built from given base functions B by composition. Suppose the composition operator is $C(f,\ g) = \lambda(x.\ f(g(x)))$. Intuitively we are defining $\mathcal{C}(N \rightarrow N) = B + Comp(\mathcal{C}(N \rightarrow N),\ \mathcal{C}(N \rightarrow N))$ where

$$Comp(S_1,\ S_2) = \{f : N \rightarrow N \mid \exists g_1 : S_1,\ \exists g_2 : S_2.\ f = C(fun(g_1), fun(g_2))\}.$$

The official definition of $\mathcal{C}(N \rightarrow N)$ is $\mu(F.\ B + Comp(F,\ F))$. Notice that if b_i are base functions, then an object like $inr(\mathcal{C}(inr(\mathcal{C}(inl(b_1),\ inl(b_2))),\ inl(b_1))$ is an element of $\mathcal{C}(N \rightarrow N)$. Given $f \in \mathcal{C}(N \rightarrow N)$, we need the mapping

$$fun : \mathcal{C}(N \rightarrow N) \rightarrow (N \rightarrow N)$$

which "pulls out" the function part of the object, e.g.

$$fun(inr(C(inr(C(inl(b_1),\ inl(b_2))),\ inl(b_1))) = C(C(b_1,\ b_2),\ b_1)$$

There is another part of the object hidden in the definition, namely the sequence of injections *inl*, *inr* and the access to B. So we can define $code(f) = inr(inr(inl(1), inl(2)),\ inl(1))$. This code tells us the structure of the definition of $fun(f)$. Based on the code we can measure the complexity of this particular *presentation* of f based on using a specific means of computing a function in $N \rightarrow N$ that is guaranteed to be equal to f.

3 Defining Computational Complexity

3.1 Meta-level complexity

A careful definition of the language, $\Pi\mu\Sigma^+$ (Promise), would be based on a class of terms. The definition would include these clauses. (It can define most of Nuprl [8].)

$$\frac{x \in var}{x \in term} \qquad \frac{t \in term}{\begin{array}{c} inl(t) \in term \\ inr(t) \in term \end{array}} \qquad \frac{x \in var,\ t \in term}{\lambda(x.\ t) \in term} \qquad \frac{a \in term\ f \in term}{ap(f,\ a) \in term}$$

$$\frac{a \in term,\ b \in term}{pair(a,\ b) \in term} \quad \frac{x \in var\ t \in term}{\mu(x.\ t) \in term} \quad \frac{t \in term,\ z \in var\ f \in term\ g \in term}{\mu{-}ind(t; z,\ f.\ g) \in term}$$

Computation is defined by structured operational semantics, for example, we have these rules among others.

$$\frac{f \downarrow \lambda(x.\ b) \quad b[a/x] \downarrow c}{ap(f,\ a) \downarrow c} \qquad \frac{g[a/z,\ \lambda(x.\ \mu{-}ind(x; z,\ f.g)/f] \downarrow c}{\mu{-}ind(a; z,\ f.\ g) \downarrow c}$$

We can define computational complexity based on this kind of scheme or on a rewrite sematnics. For example, a simple step count can be defined by

$$\frac{time(f) \downarrow n \quad time(b[a/x]) \downarrow m}{time(ap(f,\ a)) \downarrow (n + m + 1).}$$

This approach allows us to define both an evaluation function and a time-complexity function on terms. Given f a function term

$$\begin{array}{ll} eval(f) & : \{x : term \mid \exists y : termf(x) \downarrow y\} \rightarrow term \\ time(f) & : \{x : term \mid \exists y : term.\ f(x) \downarrow y\} \rightarrow \mathbb{N}. \end{array}$$

These are the *metalevel* evaluation and time-complexity functions. In some sense they give the actual complexity of terms relative to a particular implementation of the programming language.

3.2 Reflected computation

In a language with recursive types and dependent types like $\Pi\mu\Sigma^+$ it is easy to reflect the term structure and evaluation structure into the object language, building the internal type Term and the internal Eval and Time functions. This was done in detail for Nuprl [1]. We will briefly refer to these ideas later.

3.3 Computational complexity

summary of problem and a solution

The technical barrier between semantic theories and complexity theory is that semantics deals with (computable) *functions* in order to interface with conventional mathematics, and complexity theory deals with *algorithms* since costs depend on the details of the algorithm. This distinction comes down mainly to this rule for function equality used in most programming logics.

$$f =_{A \to B} g \text{ iff } \forall x : A.\ f(x) =_B g(x).$$

inductive classes of functions

Let's look at the form of inductive definitions in type theory [8, 9, 26, 14]. The ideas are similar to those in Feferman and the methods are like those of Scott [27]. If F is a monotone operation on types, say $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$, then $\mu(X.\ F)$ is a type that is essentially the least fixed point of F. This fact is characterized by equipping $\mu(X.F)$ with an induction principle. Informal notations often look like $X \overset{def}{=} F(X)$. So we might write $N \overset{def}{=} \mathbf{1} + N$ for $\mu(N.\mathbf{1} + N)$.

We are going to define a class of functions that has the form $\mu(F.\ \mathsf{Base}+\mathsf{Rec}(F)+\mathsf{Comp}(F))$ where Base is a collection of base functions and Rec and Comp define classes based on a recursion combinator and composition. We will pick Base, Rec, and Comp so that the classes defined are the polynomial time functions. The key to doing this elegantly is in the work of Leivant [22] from LICS '91; the particular result I use (at Leivant's suggestion) is from Bellantoni and Cook [3].

Let $N = \{0, 1\}$ *list*. This represents the natural numbers in the usual way (with degenerate leading 0, low order bits at the head). Let $N^0 = \mathbf{1}$, the unit type, and $N^{n+1} = N \times N^n$.

The functions we study have two kinds of numerical inputs, called *normal* and *safe* in [3]. We think of them as *canonical* inputs (or normal) and *non-canonical* (or non-normal). So they have type $N^n \times N^m \to N$. The left arguments are canonical (N^n). We use N^0 to indicate the absence of an input. The various function spaces are collected into a single domain, $\mathbb{D}$, by taking the disjoint union.

Def. $\mathbb{D} = \sum n : N.\ \sum m : N.\ N^n \times N^m \to N.$

The form of any element of $\mathbb{D}$ is $\langle n, m, f\rangle$ for f the function. The selection functions for $f \in \mathbb{D}$ is *norm*$(f) = n$, *safe*$(f) = m$, *fun*$(f) = f$.

Let $\{x : A \mid\mid B_x\}$ abbreviate the dependent product type, $\Sigma x : A.\ B_x$. The elements are $\langle a,\ b\rangle$ with $a \in A$ and $b \in B_a$. We also use the Nuprl phrase $\downarrow(P(x))$ for P a proposition. This is a proposition whose computational content has been "squashed." The official definition is $\{unit \mid P(x)\}$, so the value is $\cdot$ if $P(x)$ is true, otherwise the type is empty.

The Base functions are divided into five classes $C_0, \ldots, C_4$.

Def.

$$\begin{aligned}
C_0 = \quad & \{f : D \mid && norm(f) = 0 \\
& && \&\ safe(f) = 0 \\
& && \&\ fun(f) = \lambda p.\ 0\} \\
C_1 = \quad & \{f : D \mid && \exists n, m, i : N.\ \ norm(f) = n \\
& && \&\ safe(f) = m \\
& && \&\ fun(f) = proj(n, m, i) \\
& && \&\ \text{if } n = 0 \text{ then } i \neq 1 \\
& && \&\ \text{if } m = 0 \text{ then } i \neq n + 1\}
\end{aligned}$$

These are the projection functions where $proj(i, n, m)(x_1, \ldots, x_n; x_{n+1}, \ldots, x_{n+m}) = x_i$ provided we don't project out the element of **1**.

$C_2 = \{f : \mathbb{D} \mid \exists j : [0, 1].\ norm(f) = 0\ \&\ safe(f) = 1\ \&\ fun(f) = s_j\}$. These are the two successor functions, $s_0(\cdot; x) = 0x$ and $s_1(\cdot; x) = 1x$.

$C_3 = \{f : \mathbb{D} \mid norm(f) = 0\ \&\ safe(f) = 1\ \&\ fun(f) = pred\}$. This is the predecessor function, $pred(\cdot; 0) = 0$ and $pred(\cdot; ix) = x$.

$C_4 = \{f : \mathbb{D} \mid norm(f) = 0\ \&\ safe(f) = 3\ \&\ fun(f) = cond\}$.

This is the conditional function where

$$cond(\cdot; a, b, c) = \textit{ if } a \textit{ mod } 2 = 0 \textit{ then } b \textit{ else } c \textit{ fi}.$$

Notice that we separate the canonical arguments from the non-canonical with a semicolon.

Along with these types, we introduce constructors to build elements: $base0$ build the zero function, $\langle 0, 0, \lambda x.0\rangle$, $base2(n, m, i)$ builds the projection, $base2(i)$ builds the successor and $base3$ the predecessor and $base4$ the conditional.

Recursion is allowed "on notation," that is, we can define a function f by recursion as

$$\begin{aligned}
f(0, \overline{x}; \overline{a}) &= g(\overline{x}; \overline{a}) \\
f(iy, \overline{x}; \overline{a}) &= h_i(y, \overline{x}; f(y, \overline{x}; \overline{a}), \overline{a}).
\end{aligned}$$

We introduce the *recursion combinator* $Rec(n, m)$ to accomplish this form of recursion. Its type is

$$(N^n \times N^m \rightarrow N) \rightarrow (N^{n+1} \times N^{m+1} \rightarrow N) \rightarrow (N^{n+1} \times N^{m+1} \rightarrow N) \rightarrow (N^{n+1} \times N^m \rightarrow N).$$

The combinator reduces as follows:

$$\begin{array}{ll} Rec(n,m)(g)(h_1)(h_2)(0,\overline{x};\overline{a}) & = g(\overline{x};\overline{a}) \\ Rec(n,m)(g)(h_1)(h_2)(iy,\overline{x};\overline{a}) & = h_i(y,\overline{x};Rec(n,m)(g)(h_1)(h_2)(y,\overline{x};\overline{a}),\overline{a}). \end{array}$$

We also use a "safe composition" combinator $Comp(n,m)$ whose type is

$$(N^n \times N^m \rightarrow N) \rightarrow (N^n \times N^0 \rightarrow N)^n \rightarrow (N^n \times N^m \rightarrow N)^m \rightarrow (N^m \times N^m \rightarrow N).$$

Given $h \in N^n \times N^m \rightarrow N$, $\bar{r} \in (N^n \times N^0 \rightarrow N)^n$ and $t \in (N^n \times N^m \rightarrow N)^m$, then

$$Comp(n,m)(h)(\bar{r})(\bar{t})(\bar{x};\bar{a}) = h(\bar{r}(x;\cdot);\bar{t}(\bar{x};\bar{a}))$$

This form of composition can be used to write in combinator form a function expression, say $f(\overline{x};\overline{a})$, in terms of safe composition and projections as long as there is no subexpression $g(\overline{e}_1;\overline{e}_2)$ with a_i appearing among the canonical arguments, $\overline{e}_1$.

Using these constructs we are almost ready to define a class B which will be PTime.

$$\begin{array}{ll} \text{First let } \mathsf{Rec}(B) & = \{f : \mathbb{D} \,||\, \exists n,m : N.\ \exists g,h_1,h_2 : B.\downarrow(norm(g) = n \\ & \&\ safe(g) = m \\ & \&\ norm(h_i) = n+1 \\ & \&\ safe(h_i) = m+1 \text{ for } i = 1,1 \\ & \&\ fun(f) = Rec(n,m)(fun(g))(fun(h_1))(fun(h_2)))\} \\ \mathsf{Comp}(B) & = \{f : \mathbb{D} \,||\, \exists n,m : N.\exists h : B.\exists \overline{r} : B^n.\exists \overline{t} : B^m.\downarrow(norm(h) = n \\ & \&\ safe(h) = m \\ & \&\ \forall i : [1,n].\ (norm(r_i) = n \\ & \&\ safe(r_i) = 0) \\ & \&\ \forall j : [1,m].\ (norm(t_j) = n \ \&\ safe(t_j) = m) \\ & \&\ fun(f) = comp(n,m)(fun(h))(fun(\overline{r}))(fun(\overline{t})))\}) \end{array}$$

The functions *fun()* used in these definitions are defined to produce the function part of elements of the recursive type. This is not exactly the same as for the elements of Base since we must account now for the position of Base, Rec(B) and Comp(B) in the disjoint union, but it is a polymorphic function defined independently of the recursive type.

Among the natural constructors to associate with B are the ones for Base, now extended to B, i.e. **base0** is modified to wrap "inl" around $\langle 0\langle 0,\lambda p.0\rangle\rangle$ to give $inl(\langle 0,\langle 0,\lambda p.0\rangle\rangle)$. We also want $rec(n,m)(g)(h_1)(h_2)$ and $comp(n,m)(h)(\overline{r})(\overline{t})$ where g,h_1,h_2,h are in B and $\overline{r},\overline{t}$ are vectors of elements of B.

Def. $B = \mu(B.\ \mathsf{Base} + \mathsf{Rec}(B) + \mathsf{Comp}(B))$

Theorem (Bellantoni & Cook):

- for every $f \in PTime$, there is an $f' \in B$ such that $f(\overline{x}) = f'(\overline{x};\cdot)$.
- for every $f \in B$, $f(\overline{x}, \overline{y})$ is in PTime.

In proving the Bellantoni and Cook theorem we need to define a complexity measure on elements of B. This is easy because the constructive treatment of inductive types provides with each element of the type, say B, a code showing how it is built. For example, a constructor like

$$\begin{aligned} &\mathit{rec}(0,1)(\mathit{base}\,1(0,1,2)) \quad (\mathit{comp})(0,1)(\mathit{base}\,2(0))(\cdot)(\mathit{base}\,1(0,2,3)) \\ &\qquad\qquad\qquad\qquad\qquad\quad (\mathit{comp}(0,1)(\mathit{base}\,2(1))(\cdot)(\mathit{base}\,1(0,2,3)) \end{aligned}$$

not only builds a function in $\mathbb{D}$, but it codes up a complete description of how the element is constructed.

The equality relation on element of B is naturally defined to respect the coding of elements, so it is not extensional function equality. This has always been a feature of inductive definitions, and we now intend to exploit it.

So with each $f \in B$ we can define not only $\mathit{fun}(f)$ but a function $\mathit{code}(f)$ which is the construction history of f. We can think of these codes as an *implicit programming language* that comes with every inductive definition. It is essentially just the expression language of the constructors internalized in some natural way.

other inductive classes

The same technique used to define B can clearly be used to define the elementary functions, E, or the primitive recursive ones, Prim. For any of these classes, the codes provide a way to define resource bounds such as $\mathit{time}(f)$ or $\mathit{space}(f)$.

μ-recursive functions

It is interesting that we can use another natural feature of the Nuprl type theory to define an internal model of the *general recursive functions.* (The interpretation of this result is bound to be provocative.)

Consider the set of functions $\mathbb{F} = \Sigma n : N^+.(N^{n+1} \to N)$ and the subset $\hat{\mathbb{F}}$

$$\{f : \mathbb{F} \mid \exists n : N.\ \mathit{arity}(f) = n + 1 \&\ \forall x : N^n.\ \exists y : N. f(x,y) = 0\}.$$

On the subset we can define $\mu y.(f(x,y) = 0)$ as the least y such that $f(x,y) = 0$. In Nuprl we can just use *mu* from section 2.2.

Following Kleene [21] we can define the general recursive functions over $\mathbb{F}$, $R(\mathbb{F})$, using this clause

$$\begin{aligned} \mathsf{Mu}(C) \quad &= \{f : \mathbb{F} \mid \exists n : N\ \exists g : C.\ \mathit{arity}(f) = n + 1 \\ &\ \ \&\ \downarrow (\forall x : N^n.\ \exists y : N.\ \mathit{fun}(g)(x,y) = 0) \\ &\ \ \&\ \mathit{fun}((f) = \lambda x.\mu y(g(x,y) = 0)\} \\ \\ R(\mathbb{F}) \quad &= \mu(F.\ \mathsf{Base} + \mathsf{Prim\ Rec}(F) + \mathsf{Comp}(F) + \mathsf{Mu}(F)). \end{aligned}$$

This definition provides an implicit programming language for the general recursive functions. The operation $\downarrow (P)$ is a hiding operation which hides the proof that a y exists.

3.4 Computational complexity measures

Given a set of codes, $Code(\mathcal{C})$, for an inductive definition, we can assign a measure of resource expenditure. In general, this can be any computable function which respects the arity of the specified function, that is arity satisfies

$$fun(f) : A^{(arity(code(f)))} \rightarrow B$$

and a measure satisfies

$$M(code(f)) : A^{arity(code(f))} \rightarrow N.$$

We are only interested in measures which reflect complexity measures that can be imposed on the term structure. (Here we could require that they be Blum measures for example [4].) We want the following faithfulness condition for a measure.

Definition: A measure of computational complexity M on a class $\mathcal{C}(A^* \rightarrow B)$ is faithful iff there is a complexity measure m on term such that for all $f \in \mathcal{C}(A^* \rightarrow B)$, there is a term g such that $fun(f) =_{A^* \rightarrow B} g$ and $M(code(f))(\bar{x}) = m(g)(\bar{x})$.

This condition says that M is actually a measure, a complexity property that is definable on the terms using the evaluation relation of the implementation of the programming language and its logic.

Definition: We define the Time measure on the primitive recursive funtions, over a free algebra A with constructors c_i, say $PR(A^* \rightarrow A)$, in the usual manner [23].

Let
$U_i^n(x_1, \ldots, x_n) = x_i$

$Time(U_i^n) = 1$
$Time(h(g_1(\bar{x}), \ldots, g_n(\bar{x}))) =$
$\sum_{i=1}^{n} Time(g_i)(\bar{x}) + Time(h)(g_1(\bar{x}), \ldots, g_n(\bar{x}))$
$Time(Rg)(c_i(\bar{a}), \bar{x}) =$
$Time(g_{c_i})(Rg(a_1, \bar{x}), \ldots, Rg(a_n, \bar{x}), \bar{a}, \bar{x}) +$
$\sum_{i=1}^{n} Time(Rg)(a_i, \bar{x})$
where Rg is a brief notation for the primitive recursive function definition.

Theorem: Time is a faithful complexity measure on $\Pi\mu\Sigma^+$.

The proof is just a matter of showing that Time mimics the time measure on terms. This can be done because we have picked a class that uses only first order functions. It is more interesting to show that this can be done for inductive classes that use type 2 functions and higher. The result for type 2 can be proved using Melhorn's definitions [25, 7] or those in Bellantoni & Cook [3].

4 Resource Bounded Logics

4.1 Coding logic

As is well known, the predicate calculus can be represented in a type theory such as $\Pi\mu\Sigma^+$ using the propositions-as-types principle. The key points are that the universal quantifier $\forall x : A.\ B$ is represented as a *function space*, $\Pi x : A.\ B$ and the existential quantifier $\exists x : A.\ B$ is represented as $\Sigma x : A.\ B$.

The representation of logic allows us to state program specifications as propositions to be constructively proved. For example, the problem of finding the least prime factor of a number can be stated using $\{2 \cdot\cdot\} = \{x : N \mid 2 \leq x\}$:

$$\forall n : \{2 \cdot\cdot\}\ .\ \exists p : N.\ \ p \text{ is the least prime factor of } n$$

We would like to be able to put a *complexity constraint* on this specification, asking that p be found in polynomial time say.

4.2 Resource bounded logics

We want to define the class of feasible proofs in type theory. The major criterion is that for the types needed in number theory (specifically HA^ω), say A and B, if we feasibly prove $\forall x : A.\ \exists y : B.\ R(x, y)$, then $\exists f : Poly(A \rightarrow B).\forall x : A.R(x, f(x))$ where $Poly(A \rightarrow B)$ are the polynomial time functions from A to B.. The definition of $Poly(A \rightarrow B)$ follows the approach to the complexity of higher-order operators taken in Constable[7], Mehlhorn [25], Cook & Urquant [11].

In order to define feasible proofs, we use the propositions-as-types principle and define $\forall x : A.\exists y B.R(x, y)$ as the function type $\Pi(A; x.\Sigma(B.y.R(x, y)))$ (or in Nuprl notation, $x : A \rightarrow y : B \times R(x, y)$). We need to define a notion of polynomial time function in the dependent function class $\Pi(A; x.B)$. We follow the method of defining complexity bounds over elements of an inductively defined class large enough to include the objects we want.

numerical types

We first define the types needed for number theory (HA^ω).

$$\begin{aligned}\mathcal{T} = \mu(T.\quad & \{A : U_1 \,\|\, A = \mathbb{N} \vee A = void \vee \exists\, n, m : \mathbb{N}.A = Eq(n,m)\} + \\ & \{A : U_1 \,\|\, \exists\, B_1, B_2 : T. \downarrow (A = ty(B_1) \times ty(B_2))\} + \\ & \{A : U_1 \,\|\, \exists\, B_1, B_2 : T. \downarrow (A = ty(B_1) + ty(B_2))\} \\ & \{A : U_1 \,\|\, \exists\, B_1, B_2 : T. \downarrow (A = ty(B_1) \to ty(B_2))\} + \\ & \{A : U_1 \,\|\, \exists\, B : T. \exists P : ty(B) \to U_1. \downarrow (A = \Sigma x : ty(B).\ P(x))\} \\ & \{A : U_1 \,\|\, \exists\, B : T. \exists P : ty(B) \to U_1. \downarrow (A = \Pi x : ty(B).P(x))\})\end{aligned}$$

Notice that

$$\{A : U_1 \,|\, \exists T : \mathcal{T}.\ A = ty(T)\} \subseteq U_1.$$

Let $\mathcal{F} = \Sigma T : \mathcal{T}.\ T$; the elements are pairs $\langle T, t\rangle$ where $T \in \mathcal{T}$ and $t \in T$. Our goal is to define the subclass of "polynomial time" elements of $\mathcal{F}$, called $Poly(\mathcal{F})$. We will define this as a subtype of inductively defined class called $\mathcal{R}(\mathcal{F})$, using the measure of computational complexity for higher type objects mentioned above [7, 25, 11].

We will define $\mathcal{R}(\mathcal{F})$ to be the least class containing elements of $\mathbb{N}$ and "elements" (proofs) of the atomic types, the successor functions, the identity functions, and closed under pairing, selection, composition, constants, explicit operations on arguments (permutation of arguments, duplication of arguments and application) and primitive recursion. By the results of Grzegorczyk [17] this class will include all the primitive recursive objects of higher type. Then we define complexity functions and restrict $\mathcal{R}(\mathcal{F})$ to the polynomial time computable objects of finite type, $Poly(\mathcal{F})$. The details follow the pattern of 3.3 and are omitted here. Basically $\mathcal{R}(\mathcal{F})$ is

$$\mu(F.\ Base + Constants(F) + Pairs(F) + Explicit(F) + Recursion(F)).$$

Once we can define $Poly(\mathcal{T})$ it is possible to express the condition we needed above using propositions-as-types. For any type $T \in \mathcal{T}$, let $Poly(T)$ be those functions of $Poly(\mathcal{T})$ of type T. Then

$$\forall_{Poly}\, x : N.\ P(x) == Poly(\Pi x : N.\ P(x)).$$

In general for a complexity class $\mathcal{C}^T(\Pi x : A.\ P(x))$ we can define

$$\forall_{\mathcal{C}^T} x : A.\ P(x) = \mathcal{C}^T(\Pi x : A.\ P(x)).$$

5 Acknowledgments

I would like to thank my colleagues Stuart Allen and Chetan Murthy for discussions about the ideas of this paper and Kate Ricks for preparing the manuscript.

References

[1] S. F. Allen, R. L. Constable, D. J. Howe, and W. Aitken. The Semantics of Reflected Proof. In *Proc. of Fifth Symp. on Logic in Comp. Sci.*, pages 95–197. IEEE, June 1990.

[2] J. L. Bates and R. L. Constable. Proofs as programs. *ACM Trans. Program. Lang. and Syst.*, 7(1):53–71, 1985.

[3] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

[4] M. Blum. A machine independent theory of computational complexity. *J. ACM*, 14:322–336, 1967.

[5] S. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In *Structure in Complexity Theory, Lecture Notes in Computer Science No. 223*, pages 77–103. Springer-Verlag, 1986.

[6] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proc. of the 1964 Int. Congress for Logic, Methodology, and Phil. of Sci.*, pages 24–30, North-Holland, 1965.

[7] R. L. Constable. Type two computational complexity. In *Proc. 5th ACM Symp., Theory of Computing*, pages 108–121, 1973.

[8] R. L. Constable, S. F. Allen, H. Bromley, W. Cleaveland, J. Cremer, R. Harper, D. J. Howe, T. Knoblock, N. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics with the Nuprl Development System.* Prentice-Hall, NJ, 1986.

[9] R. L. Constable and N. Mendler. Recursive Definitions in Type Theory. In *Proc. of Logics of Prog. Conf.*, pages 61–78, January 1985. (Cornell TR 85-659).

[10] R. L. Constable and D. Zlatin. The type theory of PL/CV3". *ACM Trans. on Prog. Lang. and Systems*, 6(1):94–117, January, 1984.

[11] S. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. In *Proceedings of 21st Symposium on the Theory of Computing*, pages 107–112. ACM, 1989. To appear in *Annals of Pure and Applied Logic.*

[12] S. A. Cook and B. M. Kapron. Characterizations of the basic feasible functionals of finite type. In S. Buss and P. Scott, editors, *Proceedings of MSI Workshop on Feasible Mathematics*, pages 71–95, New York, 1990. Birkhauser-Boston.

[13] N. G. deBruijn. The mathematical language Automath, its usage and some of its extensions. In *Symp. on Automatic Demonstration,* Lecture Notes in Mathematics, Vol. 125, pages 29–61. Springer-Verlag, 1970.

[14] P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

[15] S. Feferman. A language and axioms for explicit mathematics. In *Algebra and Logic,* Lecture Notes in Mathematics, pages 87–139. Springer-Verlag, 1975.

[16] J.-Y. Girard, A. Scedrov, and P. J. Scott. Bounded linear logic. *Theoretical Computer Science*, 97(1):1–66, 1992.

[17] A. Grzegorczyk. Recursive objects in all finite types. *Fundamenta Mathematicae*, 54:73–93, 1964.

[18] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematics Society*, 117:285–306, 1965.

[19] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, NY, 1980.

[20] D. J. Howe. On computational open-endedness in Martin-Löf's type theory. In *Proc. of Sixth Symp. on Logic in Comp. Sci.*, pages 162–172. IEEE Computer Society, 1991.

[21] S. C. Kleene. *Introduction to Metamathematics.* D. Van Nostrand, Princeton, 1952.

[22] D. Leivant. Functions over free algebras definable in the simply typed lambda calculus. *Theoretical Computer Science*, 121:309–321, 1993.

[23] D. Leivant. Stratified functional programs and computational complexity. In *Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 325–333, Charleston, SC, January 1993. ACM, ACM Press.

[24] P. Martin-Löf. *Intuitionistic Type Theory, Studies in Proof Theory, Lecture Notes.* Bibliopolis, Napoli, 1984.

[25] K. Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Sciences*, pages 148–176, 1976.

[26] F. Pfenning and C. Paulin-Mohring. Inductively defined types in the calculus of constructions. In *Mathematical Foundations of Program Semantics, 5th International Conference,* Lecture Notes in Computer Science, Vol. 442, pages 209–228. Springer-Verlag, 1989.

[27] D. Scott. Data types as lattices. *SIAM J. Comput.*, 5:522–87, 1976.

[28] S. Smith and R. L. Constable. Partial objects in constructive type theory. In *Proc. of Second Symp. on Logic in Comp. Sci.*, pages 183–93. IEEE, Washington, D.C., 1987.

[29] K. Weihrauch. A simple and powerful approach for studying constructivity, computability and complexity. In J. Myers and M. O'Donnell, editors, *Constructivity in Computer Science, Logic in Computer Science 613*, pages 228–246. Springer-Verlag, 1991.

Light Linear Logic

Jean-Yves Girard
Laboratoire de Mathématiques Discrètes
UPR 9016 – **CNRS**
163, Avenue de Luminy, Case 930
F-13288 Marseille Cedex 09

girard@lmd.univ-mrs.fr

Abstract

The abuse of structural rules may have damaging complexity effects.

1 INTRODUCTION : A LOGIC OF POLYTIME ?

We are seeking a « logic of polytime ». Not yet one more axiomatization, but an intrinsically polytime system. Our methodological bias will be to consider that the expressive power of a system is the complexity of its cut-elimination procedure, and we therefore seek a system with a polytime complexity for cut-elimination (to be precise : besides the *size* of the proof, there will be an auxiliary parameter, the *depth*, controlling the degree of the polynomial). This cannot be achieved within classical or intuitionistic logics, because of structural rules, especially contraction : this is why the complexity of cut-elimination in all extant logical systems (including the standard version of linear logic which controls structural rules without forbidding them) is catastrophic, elementary (towers of exponentials) or worse. Light Linear Logic is a purely logical system with a more careful handling of structural rules : this system is strong enough to represent all polytime functions, but cut-elimination is (locally) polytime. With **LLL** our control over the complexity of cut-elimination improves a lot.

But this is not the only potentiality of **LLL** : why not transforming it into a system of mathematics, and try to formalize « polytime mathematics » in the same way as Heyting arithmetic formalizes constructive mathematics ? The possibility is clearly open, since **LLL** admits extensions into a naive set-theory, with full comprehension, still with a polytime cut-elimination. This system admits full induction on data types, which shows that, within **LLL**, induction is compatible with low complexity...

1.1 Background

1.1.1 Complexity of normalization

Our goal is to find a logical system in which the I/O dependencies are given by polytime functions. We shall try a proof-theoretic approach, namely to make sure that cut-elimination is polytime. In fact we shall concentrate on the following question : which *logical* system(s) induce normalizations of a given complexity (polytime or not) ?

There is an answer, namely **MALLq** (multiplicative-additive-quantifiers (of any order) linear logic) : the small normalization theorem of [1] assigns a size bound to proofs ; this size shrinks during *lazy* cut-elimination, hence induces linear time functions. A crucial technological point is that the notion of cut-degree disappears, i.e. the procedure is not dependent of the fact that cuts are replaced with simpler ones. The operation has perfectly succeeded, but the patient died (of starvation) : this system is desperately inexpressive.

Linear logic wisely has another stock of connectives, namely *exponentials* which should compensate for this limitation, by restoring the necessary amount of structural manipulations (mainly the contraction rule). Now the patient is still dying, but of overfeeding : the complexity is no longer bounded by any reasonable measure, since usual logic (classical or intuitionistic) imbeds. The question is therefore to find more reasonable connectives, sitting in between **MALLq** and **LL**. These connectives are the *light* exponentials.

The first attempt dates back from 1987 (joint work with A. Scedrov & P. Scott, [3]) and is based on the idea of replacing $!A$ (which usually means A *ad libitum*) by $(1 \& A) \otimes \ldots \otimes (1 \& A)$, i.e. essentially by a finite tensor power of A, $!_n A$. It is also immediate to see that the rules of weakening, dereliction, contraction and promotion are still valid w.r.t. bounded exponentials : the bounds are respectively given by $0, 1, +, .$, i.e. the maintenance is polynomial. This very good starting point leads to **BLL** (Bounded Linear Logic) ; **BLL** has a lot of qualities (it exactly corresponds to polytime etc.), but it has a major drawback : it mentions the polynomial bounds who should remain hidden [1]. By the way observe that **BLL** is far from giving good bounds : the main property of exponentials is the isomorphism between $!A \otimes !B$ and $!(A \& B)$, but **BLL** yields the bounds $!_{n+m}(A \& B) \multimap !_n A \otimes !_m B$ and $!_n A \otimes !_n B \multimap !_n(A \& B)$, which induce by composition $!_{2n}(A \& B) \multimap !_n(A \& B)$, not quite an isomorphism.

Since this first attempt, many other restrictions have been tried by Danos, Joinet, Lafont, Schellinx and myself, without obtaining truly convincing results [2].

Other connections between polytime complexity and normalization have been made in the recent years, typically works of Leivant, Leivant-Marion [7, 8], and Hillebrand-Kanellakis-Mairson, [5]. These approaches stay inside typed λ-calculi, i.e. systems which are by no standards polytime (the complexity is at least elementary), but they individualize certain interesting situations where the complexity is exactly polytime (this is based on the fact that in traditional sit-

[1] These bounds do not refer to time, but to size.

[2] *A posteriori* it is clear that all these attempts failed because they included the principle [V], see 1.1.3.

uations, the complexity is determined by the cut-formulas : the basic idea is to restrict one to cuts of a certain form to achieve complexity effects). The obvious advantage of these approaches lies is the use of traditional systems (or at least systems not too far from that). But these systems can hardly claim to bring some insight as to the logical nature of polytime, since as soon as we iterate their logical primitives, the complexity explodes, in other terms the logical primitives (basically intuitionistic implication) make mistakes w.r.t. complexity. To take an analogy : classical (Peano) arithmetic is indeed constructive for Π^0_2 sentences, where it coincides with Heyting arithmetic ; for more complex formulas, it is constructively wrong. There is therefore still a want for a system which is intrinsically polytime, in the way that Heyting arithmetic is intrinsically constructive. An answer is **LLL**.

1.1.2 LLL and naive set-theory

We are seeking a logical system with a light complexity. This basically means that the cut-elimination bounds will not depend on the cut-formulas, i.e. will not rely on the replacement of a cut by a simpler one. Then such a system will accommodate naive set-theory. This is simply because naive set-theory has a normalization procedure (the one described by Prawitz in the 60's) which will terminate in such a framework. Typically this works for **MALLq** (this is indeed an old remark of Grishin, [4]), since the naive comprehension scheme does not prevent normalization from shrinking !

Our crucial test for selecting the right rules, will be to check whether or not naive set-theory becomes inconsistent with the proposed set of rules for exponentials. Typically, naive set-theory enables us to get fixpoints of any logical operation (like naive function theory, i.e. λ-calculus), and it suffices to check the impossibility of getting a contradiction from fixpoint. The best candidate is the one arising from Russell's paradox, i.e. $A \simeq !A^\perp$. For those who find this methodology surprising, we can phrase it differently : inconsistency (i.e. failure of cut-elimination) is the case of a cut-elimination that does not terminate, and this is due to the same phenomenon as heavy complexity, i.e. cut-elimination that does not terminate within "feasible" bounds.

We shall therefore tailor our light exponentials w.r.t. naive set-theory, but keep only the second order propositional logic arising from this study. It would be possible to do much more : one can add the logical rules of naive set-theory to **LLL**, and this provides a very powerful system. In this system extensionality fails (as already observed by Grishin), but Leibniz equality can do wonders. Integers in unary (or binary) numeration can be defined, and full induction therefore works. In other terms, one can get a pure logical system without any proper axiom, which contains both a light set-theory and a light arithmetic.

1.1.3 Dissection of exponentials

Exponentials are used to "classicize" **LL**. This involves a lot of micro properties, that we can individualize below :

▸ [I] : $!(A \,\&\, B) \multimap !A \otimes !B$ (and $!\top \multimap 1$)

- [II] : $!A \otimes !B \multimap !(A \,\&\, B)$ (and $1 \multimap !\top$)

- [III] : from $A \multimap B$ derive $!A \multimap !B$

- [IV] : $!A \multimap ?A$

- [V] : $!A \otimes !B \multimap !(A \otimes B)$ (and $!1$)

- [VI] : $!A \multimap A$

- [VII] : $!A \multimap !!A$

The first two principles express the usual isomorphism which is responsible for the name "exponential" : principle [I] expresses contraction and principle [II] expresses weakening.
Principle [III] expresses functoriality of the exponentials and is absolutely basic.
Principle [IV] is a weak form of dereliction (i.e. principle [VI]).
These four principles will constitute the basis of **LLL**.
Principle [V] enables one to give a multilinear version of functoriality (from $\Gamma \vdash B$ derive $!\Gamma \vdash !B$), and will not be accepted in **LLL**, although it is also compatible with naive set-theory [3].
In presence of fixpoint $A \simeq !A^{\perp}$, it is possible to derive the sequent $\vdash$ (so no cut-elimination !), in two ways

- from [I] + [III] + [VI]

- from [I] + [III] + [IV] + [VII]

In both cases one first proves $\vdash ?A, ?A$ from the fixpoint principle $\vdash A, ?A$:

- dereliction [VI] yields $\vdash ?A, ?A$

- [III] yields $\vdash !A, ??A$, then [IV] yields $\vdash ?A, ??A$ and [VI] removes the extra "?"

From $\vdash ?A, ?A$ contraction [I] yields $\vdash ?A$, and by fixpoint one gets $\vdash A^{\perp}$, which by promotion yields in turn $\vdash !A^{\perp}$. We end with a cut between $\vdash ?A$ and $\vdash !A^{\perp}$.
Therefore principles [VI] and [VII] are definitely excluded.
The failure of dereliction is the reason for the introduction of the weaker principle [IV]. Unfortunately it turns out that this principle is too weak in terms of expressive power, and this is the reason why an additional modality is introduced.

1.1.4 The three modalities

In **LLL** there are indeed three modalities $!, \S, ?$. $\S$ (*neutral*) is a new intermediate modality. $\S$ is self-dual, i.e. $(\S A)^{\perp}$ is $\S A^{\perp}$, and its intuitive meaning is the (common) unary case of ! and ?. The principles of **LLL** are :

- [I], [II], [III] (written in terms of !, ?)

[3] It yields another logic with an elementary complexity for cut-elimination, **ELL**.

- from $A \vdash B$ derive $\S A \vdash \S B$ [VIII]
- $!A \vdash \S A$ [IX]
- $\S A \otimes \S B \vdash \S(A \otimes B)$ (and $\S 1$) [X]

[VIII] is just usual functoriality, and [X] enables one to get a n-ary version ; [IX] is a compensation for the want of dereliction...observe that it implies by duality $\S A \vdash ?A$ and is therefore an improved version of [IV].
These principles can be organized along a sequent calculus which enjoys a cut-elimination with polynomial bounds, as we shall see below.

1.2 Expressive power of LLL

LLL can be seen as a system of set-theory (or arithmetic). It can also be seen as a system of typed λ-calculus. We have to explain how to encode data and polytime algorithms.

1.2.1 Integers

Remember that complexity depends on the representation of data : typically integers can be given in tally representation or in binary representation, with an exponential reduction of their size. This is why we introduce two types, **int** and **bint** for integers.
Tally integers can be given the type $\mathbf{int} = \forall X.!(X \multimap X) \multimap \S(X \multimap X)$, where X is a second order variable. The traditional type $\forall X.!(X \multimap X) \multimap (X \multimap X)$ cannot be used for want of dereliction, and the immediate substitute for it, $\forall X.!(X \multimap X) \multimap !(X \multimap X)$ cannot be used either, since the principle $!(A \multimap A); !(A \multimap A) \vdash !(A \multimap A)$ is not part of **LLL**. Observe that *addition* can be given the type $\mathbf{int}; \mathbf{int} \vdash \mathbf{int}$ and *multiplication* can be given the type $\mathbf{int}; !\mathbf{int} \vdash \S\mathbf{int}$. In fact any polynomial P in k variables can be given [4] a type $\mathbf{int} \otimes \ldots \otimes \mathbf{int} \multimap \S^k\mathbf{int}$, where k is an integer depending on the degree of P. Typically, x^2 can be given the type $\mathbf{int} \multimap \S\S\mathbf{int}$.
Binary integers (lists of 0 and 1) can be given the type $\mathbf{bint} = \forall X.!(X \multimap X) \otimes !(X \multimap X) \multimap \S(X \multimap X)$. There is a canonical map which consists in replacing a binary list with a unary one, i.e. $\sharp(x)$ is the length of x in tally representation. The type of this map is $\mathbf{bint} \multimap \mathbf{int}$.

1.2.2 Turing machines

Let us fix the alphabet and the set of states of our Turing machines. In order to represent our machines, all we now have to do is to find a type **Tur**, in such a way that configurations (tape + state) of such a machine are exactly the objects of type **Tur**. **Tur** must also be such that the instructions of a Turing machine induce objects of type $\mathbf{Tur} \multimap \mathbf{Tur}$. Several possibilities are at hand,

[4] Up to minor details

but the simplest is for sure to use the fixpoint facility coming from the naive comprehension axiom. (The fixpoint of the operator $\Phi[p]$ is obviously $t \in t$, with $t = \{x \mid \Phi[x \in x]\}$).

1.2.3 Polytime functions

Let us now take a polytime program from binary integers to binary integers, with runtime P. We can consider the function of type $\mathbf{bint} \multimap \S\mathbf{Tur}$ which yields the input configuration of the machine, as well as the function of type $\mathbf{bint} \multimap (\S)^k int$ which yields the number of steps ; if our program is represented by φ of type $\mathbf{Tur} \multimap \mathbf{Tur}$, then we eventually get an object of type $\mathbf{bint} \multimap (\S)^{k+2}\mathbf{Tur}$ which yields, in function of the binary input, the output tape.

This representation has no pretension to elegance : its only virtue is to show the expressive power of **LLL**. Since **LLL** is a real logical system in which it is impossible to be worse than polytime, smarter representations must be found.

1.3 Cut-elimination

The sequent calculus naturally associated with **LLL** is a double-layer version, i.e. with additive and multiplicative disjunctions. This is not very friendly [5], but after all sequent calculus is not the only proof-theoretic syntax, and one can use more sophisticated technologies, typically proof-nets. The proof-net technology has made essential progress in the recent years, and it now possible to represent the full sequent calculus, in terms of proof-nets with boxes. Boxes are only needed for exponentials : the promotion rule [III] induces a box as well as the dereliction (or rather its weaker version) (combination of [VIII],[IX],[X]). The main parameters of a proof are

- its *depth*, which is the nesting number of the exponential boxes ;
- its *size* which counts the number of links ;
- its partial sizes, i.e. the size of the part of the net which is at a certain depth.

Cut-elimination works as follows : it is a lazy one, i.e. no cut is eliminated « inside a &-box » [6], which is performed layer after layer, first starting with depth 0. After eliminating the cuts of depth 0, the sizes (which were $s_0, s_1, s_2, \ldots$) become at most $s_0, s_0s_1, s_0s_2, \ldots$ From this it is immediate that after eliminating all cuts, the final size is roughly s^{2^d}, where s, d are the original size and depth. What makes the argument work is that the light rules are of constant depth, i.e. that no change (increase or decrease) may happen during cut-elimination. By the way, these bounds are the simplest refutation for additional principles

[5] The proof of cut-elimination with the polynomial bounds is not manageable with sequent calculus.

[6] Technically this is the notion of ready cut coming from [2]... strictly speaking there are no longer additive boxes.

such as $\S(A \oplus B) \vdash \S A \oplus \S B$: a function of type $\mathbf{bint} \vdash \S(1 \oplus 1)$ computes in quadratic time (since we can stop the computation when the $\oplus$-rule occurs, whereas the type $\mathbf{bint} \vdash \S 1 \oplus \S 1$ requires only linear time.
With proof-nets, it is easy to see that the bound immediately yields a $s^{2^{d+2}}$ time bound for usual I/O, like binary strings. Moreover a binary string is represented by a proof-net of depth 1, hence the application $f(s)$ of a given function f of type $\mathbf{bint} \multimap (\S)^k \mathbf{Tur}$ to a binary string s will have the same depth as f, i.e. the computation will run in a time which is polynomial in the size of s.

2 THE SYNTAX OF LLL

Constructive logic is basically propositional ; this is why we focus on (second-order) propositional **LLL**. However the system is quite flexible and accepts quantifiers of any order, including set-quantifiers.

2.1 The formulas of LLL

LLL has the same connectives as usual linear logic but for the exponentials : there is an extra (self-dual) modality, § (*neutral*) is added [7].

Definition 1
Literals (T) and formulas (F) are defined as follows

$T = \alpha, \beta, \gamma \ldots, \alpha^\perp, \beta^\perp, \gamma^\perp \ldots$

$F = T, 1, \perp, 0, \top, !F, \S F, ?F, F \otimes F, F \parr F, F \& F, F \oplus F, \forall \alpha F, \exists \alpha F$

Definition 2
(Linear) negation is a defined connective :

- $(\alpha)^\perp = \alpha^\perp, (\alpha^\perp)^\perp = \alpha$
- $1^\perp = \perp, \perp^\perp = 1$
- $0^\perp = \top, \top^\perp = 0$
- $(!A)^\perp = ?A^\perp, (\S A)^\perp = \S A^\perp, (?A)^\perp = !A^\perp$
- $(A \otimes B)^\perp = A^\perp \parr B^\perp, (A \parr B)^\perp = A^\perp \otimes B^\perp$
- $(A \& B)^\perp = A^\perp \oplus B^\perp, (A \oplus B)^\perp = A^\perp \& B^\perp$
- $(\forall \alpha A)^\perp = \exists \alpha A^\perp, (\exists \alpha A)^\perp = \forall \alpha A^\perp$

Linear implication is a defined connective :
$A \multimap B = A^\perp \parr B$

[7] We have been tempted to replace $!, \S, ?$ by the musical symbols ♯, ♮, ♭

2.2 The sequents of LLL

Definition 3

- *A* discharged *formula is an expression* $[A]$, *where* A *is a formula ;*
- *A* block $\mathbf{A}$ *is a sequence* $A_1, \ldots, A_n$ *of formulas, or a single discharged formula* $[A]$ *; the standard case is that of a block of length* 1, *for which we use the notation* A *or* $[A]$ *;*
- *A* sequent *is an expression* $\vdash \mathbf{A_1}; \ldots; \mathbf{A_n}$, *where* $\mathbf{A_1}, \ldots, \mathbf{A_n}$ *are either discharged formulas or blocks. The standard case is that of a sequence of (undischarged) formulas ; even more standard is the case when the sequence consists of exactly one formula.*

Remark. —

- A *block* $A_1, \ldots, A_n$ is hypocrisy for the formula $A_1 \oplus \ldots \oplus A_n$;
- A discharged formula $[A]$ is hypocrisy for $?A$;
- If $\mathbf{A_1}, \ldots, \mathbf{An}$ are hypocrisy for formulas $A_1, \ldots, A_n$, then the sequent $\vdash \mathbf{A_1}; \ldots; \mathbf{An}$ is hypocrisy for the formula $A_1 \parr \ldots \parr A_n$.

2.3 The sequent calculus of LLL

Identity / Negation

$$\frac{}{\vdash A; A^{\perp}}\ (identity) \qquad \frac{\vdash \Gamma; A \quad \vdash A^{\perp}; \Delta}{\vdash \Gamma; \Delta}\ (cut)$$

Structure

$$\frac{\vdash \Gamma; \mathbf{A}; \mathbf{B}; \Delta}{\vdash \Gamma; \mathbf{B}; \mathbf{A}; \Delta}\ (M\text{-}exchange) \qquad \frac{\vdash \Gamma; \mathbf{A}, C, D, \mathbf{B}}{\vdash \Gamma; \mathbf{A}, D, C, \mathbf{B}}\ (A\text{-}exchange)$$

$$\frac{\vdash \Gamma}{\vdash \Gamma; [A]}\ (M\text{-}weakening) \qquad \frac{\vdash \Gamma; \mathbf{A}}{\vdash \Gamma; \mathbf{A}, B}\ (A\text{-}weakening)$$

$$\frac{\vdash \Gamma; [A]; [A]}{\vdash \Gamma; [A]}\ (M\text{-}contraction) \qquad \frac{\vdash \Gamma; \mathbf{A}, B, B}{\vdash \Gamma; \mathbf{A}, B}\ (A\text{-}contraction)$$

Logic

$$\frac{}{\vdash 1}\ (one) \qquad \frac{\vdash \Gamma}{\vdash \Gamma; \perp}\ (false)$$

$$\frac{\vdash \Gamma; A \quad \vdash B; \Delta}{\vdash \Gamma; A \otimes B; \Delta}\ (times) \qquad \frac{\vdash \Gamma; A; B}{\vdash \Gamma; A \parr B}\ (par)$$

$$\frac{}{\vdash \Gamma; \top}\ (\textit{true}) \qquad\qquad (\textit{no rule for zero})$$

$$\frac{\vdash \Gamma; A \quad \vdash \Gamma; B}{\vdash \Gamma; A \,\&\, B}\ (\textit{with}) \qquad \frac{\vdash \Gamma; A}{\vdash \Gamma; A \oplus B}\ (\textit{left plus}) \qquad \frac{\vdash \Gamma; B}{\vdash \Gamma; A \oplus B}\ (\textit{right plus})$$

$$\frac{\vdash B_1, \ldots, B_n; A}{\vdash [B_1]; \ldots; [B_n]; !A}\ (\textit{of course}) \qquad \frac{\vdash \Gamma; [A]}{\vdash \Gamma; ?A}\ (\textit{why not})$$

$$\frac{\vdash B_1 \mid \ldots \mid B_n; A_1; \ldots; A_m}{\vdash [B_1]; \ldots; [B_n]; \S A_1; \ldots; \S A_m}\ (\textit{neutral : } B_1, \ldots, B_n \textit{ are formulas separated by commas or semicolons})$$

$$\frac{\vdash \Gamma; A}{\vdash \Gamma; \forall \alpha\, A}\ (\textit{for all : } \alpha \textit{ is not free in } \Gamma) \qquad \frac{\vdash \Gamma; A[B/\alpha]}{\vdash \Gamma; \exists \alpha\, A}\ (\textit{there is})$$

2.4 The expressive power of LLL

Our goal here is to prove that polytime functions can be represented in **LLL**. This can be established by various means. We adopt the simplest (maybe not the most elegant) solution, namely to encode polytime Turing machines in an intuitionistic version of **LLL**, **ILLL**. There will be a forgetful function of **ILLL** into system $\mathcal{F}$ (with conjunction), hence the ultimate interpretation will be in system $\mathcal{F}$, in which the representation of data, algorithms, is quite familiar...

2.4.1 The system ILL

The language of **ILL** is based on the connectives $\otimes, \&, \multimap, !, \S$, and second-order quantification. The sequents of **ILL** are of the form $\mathbf{A_1}; \ldots; \mathbf{A_n} \vdash B$, where $\mathbf{A_1}, \ldots; \mathbf{A_n}$ are blocks, and B is a formula. The formulas $1^k = !^k 1$ are allowed in the blocks, although they are not part of the language of **ILLL** [8]. The rules of **ILLL** are those that remain correct when we translate $\mathbf{A_1}; \ldots; \mathbf{A_n} \vdash B$ as $\vdash \mathbf{A_1}^\perp; \ldots; \mathbf{A_n}^\perp; B$.

The forgetful functor (*erasure*) from **ILLL** into $\mathcal{F}$ is defined as follows :

- To a formula A of **ILLL** we associate A^-, a type of $\mathcal{F}$, as follows : ! and § are erased, $\otimes$ and & are replaced with $\wedge$, $\multimap$ is replaced with $\Rightarrow$, variables and quantifiers are unchanged ;

- To a sequent S of **ILLL** we associate a sequent S^- of second order propositional logic :

[8] This is due to the technical restrictions on the !-rule ; should **ILLL** be developed for its own sake, these special formulas should be replaced with markers...

- In S remove all discharges $[\cdot]$;
- In S replace all semicolons by commas ;
- In S replace all formulas of **LLL** by their erasure ;
- In S remove all formulas 1^k.

Typically the erasure of $A, B, !!1; [1]; [C]; D, E \vdash F$ will be interpreted as $A^-, B^-, C^-, D^-, E^- \vdash F^-$.

▸ To a proof $\mathbf{\Pi}$ of S we associate a proof $\mathbf{\Pi}^-$ of S^-, or equivalently a term of system $\mathcal{F}$. Typically a proof of $A, B, !!1; [1]; [C]; D, E \vdash F$ will induce a term t, depending on variables v, w, x, y, z :

$$v\ : A^-, w\ : B^-, x\ : C^-, y\ : D^-, z\ : E^- \vdash t\ : F^-$$

The interpretation is straightforward ; observe that the erasure of 1^k is unproblematic, since 1^k indeed deals with weakening, i.e. dummy variables.

The basic idea behind the erasure is that (intuitionistic) linear logic (light or not) can be viewed as a more refined way to speak of implication, conjunction, erasing reuse. These refinements are not taken into account in intuitionistic logic, and the forgetful functor collapses the two conjunctions, ignores exponentials (and therefore destroys 1^k, a very subtle handling of weakening). This is reflected in the translation of the formulas and also of the sequents, where the additive, multiplicative and exponential layers (represented by ",", ";", $[\cdots]$ are collapsed into a comma. When we shall represent data and algorithms in **ILLL**, we shall implicitly refer to their forgetful image in $\mathcal{F}$. It goes without saying that the notion of reduction to be defined in **LLL** is compatible with the notion of reduction of $\mathcal{F}$, so that the only consideration of the forgetful images matters. In what follows the most important functions are represented in details ; we assume that the reader is most familiar with system $\mathcal{F}$, the Curry-Howard isomorphism which identifies natural deduction with typed λ-terms, and therefore that he has no problem to synthesize the λ-term associated with a proof in second-order intuitionistic sequent calculus.

2.4.2 Representation of tally integers

Integers We define the type **int** of tally integers by

$$\mathbf{int} = \forall\alpha.\ !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$$

The tally integer $\overline{n}$ is obtained as follows :

$$\begin{array}{c}
\alpha \vdash \alpha \quad \alpha \vdash \alpha \\ \hline
\alpha \multimap \alpha ; \alpha \vdash \alpha \quad \alpha \vdash \alpha \\ \hline
\alpha \multimap \alpha ; \alpha \multimap \alpha ; \alpha \vdash \alpha \\
\vdots \\
\alpha \multimap \alpha ; \ldots ; \alpha \multimap \alpha ; \alpha \vdash \alpha \\ \hline
\alpha \multimap \alpha ; \ldots ; \alpha \multimap \alpha \vdash \alpha \multimap \alpha \\ \hline
[\alpha \multimap \alpha] ; \ldots ; [\alpha \multimap \alpha] \vdash \S(\alpha \multimap \alpha) \\ \hline\hline
[\alpha \multimap \alpha] \vdash \S(\alpha \multimap \alpha) \\ \hline
!(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha) \\ \hline
\vdash !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) \\ \hline
\vdash \forall \alpha .\, !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)
\end{array}$$

It is immediate that $\mathbf{int}^- = \forall\alpha.\ (\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha)$, and that $(\overline{n})^- = \Lambda\alpha.\lambda x^{\alpha\Rightarrow\alpha}.\lambda y^{\alpha}.x(x\ldots(x(y))\ldots)$.

Addition

Addition is proof $+$ of $\mathbf{int}, \mathbf{int} \vdash \mathbf{int}$ obtained as follows :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha \vdash \alpha \quad \alpha \vdash \alpha}{\alpha \multimap \alpha ; \alpha \vdash \alpha} \quad \alpha \vdash \alpha}{\alpha \multimap \alpha ; \alpha \multimap \alpha ; \alpha \vdash \alpha}}{\alpha \multimap \alpha ; \alpha \multimap \alpha \vdash \alpha \multimap \alpha}}{\S(\alpha \multimap \alpha) ; \S(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)} \quad \dfrac{\alpha \multimap \alpha \vdash \alpha \multimap \alpha}{[\alpha \multimap \alpha] \vdash !(\alpha \multimap \alpha)}}{\mathbf{int}_\alpha ; \S(\alpha \multimap \alpha) ; [\alpha \multimap \alpha] \vdash \S(\alpha \multimap \alpha)} \quad \dfrac{\alpha \multimap \alpha \vdash \alpha \multimap \alpha}{[\alpha \multimap \alpha] \vdash !(\alpha \multimap \alpha)}}{\mathbf{int}_\alpha ; \mathbf{int}_\alpha ; [\alpha \multimap \alpha] ; [\alpha \multimap \alpha] \vdash \S(\alpha \multimap \alpha)}}{\mathbf{int}_\alpha ; \mathbf{int}_\alpha ; [\alpha \multimap \alpha] \vdash \S(\alpha \multimap \alpha)}}{\mathbf{int}_\alpha ; \mathbf{int}_\alpha \vdash \mathbf{int}_\alpha}}{\mathbf{int} ; \mathbf{int}_\alpha \vdash \mathbf{int}_\alpha}}{\mathbf{int} ; \mathbf{int} \vdash \mathbf{int}_\alpha}}{\mathbf{int} ; \mathbf{int} \vdash \mathbf{int}}$$

with $\mathbf{int}_\alpha = !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$. It is immediate that the erasure of $+$ is the usual representation of addition in $\mathcal{F}$.

Multiplication

Multiplication is a proof × of $!\mathbf{int};\mathbf{int} \vdash \S\mathbf{int}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{int} \vdash \mathbf{int} \qquad \begin{matrix}\vdots\,\overline{0}\\ \vdash \mathbf{int}\end{matrix}}{\mathbf{int} \multimap \mathbf{int} \vdash \mathbf{int}}}{\S(\mathbf{int} \multimap \mathbf{int}) \vdash \S\mathbf{int}} \qquad \dfrac{\dfrac{\begin{matrix}\vdots\,+\\ \mathbf{int};\mathbf{int} \vdash \mathbf{int}\end{matrix}}{\mathbf{int} \vdash \mathbf{int} \multimap \mathbf{int}}}{[\mathbf{int}] \vdash !(\mathbf{int} \multimap \mathbf{int})}}{[\mathbf{int}];\mathbf{int}_{\mathbf{int}} \vdash \S\mathbf{int}}}{[\mathbf{int}];\mathbf{int} \vdash \S\mathbf{int}}}{!\mathbf{int};\mathbf{int} \vdash \S\mathbf{int}}$$

It is immediate that the erasure of × is the usual representation of multiplication in $\mathcal{F}$.

Iteration

The principle of *iteration* is derivable : if Γ is a block and not discharged, then from a proof of $\Gamma \vdash A \multimap A$ and a proof of $\Delta \vdash \S A$, one can derive $[\Gamma];\Delta;\mathbf{int} \vdash \S A$:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A \multimap A}{[\Gamma] \vdash !(A \multimap A)} \qquad \dfrac{\S A \vdash \S A \quad \Delta \vdash \S A}{\Delta;\S A \multimap \S A \vdash \S A}}{[\Gamma];\Delta;\mathbf{int}_A \vdash \S A}}{[\Gamma];\Delta;\mathbf{int} \vdash \S A}$$

It is immediate that the erasure of iteration is the usual representation of iteration in $\mathcal{F}$; however, very few actual iterations of $\mathcal{F}$ can be obtained this way. The types $\mathbf{list}^{\mathbf{k}}$, to be defined below, have similar primitives, including a notion of iteration.

Coercions

Observe that any sequent $1^{k_1};\ldots;1^{k^p};\Gamma \vdash A$ can be replaced with $1^k;\Gamma \vdash A$, provided $k \geqslant k^1,\ldots,k_p$. We shall content ourselves with a weaker typing of integers, namely $1^p \vdash \S^q\mathbf{int}$ (in general $p = q$, but we do not mind about the actual value of p. A n-ary function from integers to integers will be given a type $1^p;\mathbf{int};\ldots;\mathbf{int} \vdash \S^q\mathbf{int}$.

The successor function is naturally typed $\vdash \mathbf{int} \vdash \mathbf{int}$, which can be replaced with $!^k\mathbf{int} \vdash !^k\mathbf{int}$, and therefore by $1 \vdash !^k\mathbf{int} \multimap 1^k\mathbf{int}$. The integer $\overline{0}$ can be given the type $1 \vdash \mathbf{int}$, hence the type $1^k \vdash !^k\mathbf{int}$, and also the type
$1^{k+1} \vdash \S!^k\mathbf{int}$. We are in position to apply iteration and we get a function which is typed $[1];1^{k+1};\mathbf{int} \vdash \S!^k\mathbf{int}$, which can be replaced with $1^{k+1};\mathbf{int} \vdash \S!^k\mathbf{int}$. This function is essentially the identity on integers, but it changes the type, and we call it a *coercion*.

In a similar way, we can define coercions of type $1^{p+q}; \mathbf{int} \vdash \S^p !^q \mathbf{int}$, when $p \neq 0$. An immediate consequence is that the multiplication can be given a more even type : replace $!\mathbf{int}; \mathbf{int} \vdash \S\mathbf{int}$ with $\S!\mathbf{int}; \S\mathbf{int} \vdash \S^2\mathbf{int}$, and compose with the coercions $1^2; \mathbf{int} \vdash \S!\mathbf{int}$ and $!1; \mathbf{int} \vdash \S\mathbf{int}$, in order to get $1^2; !1; \mathbf{int}; \mathbf{int} \vdash \S^2\mathbf{int}$, which can be simplified into $1^2; \mathbf{int}; \mathbf{int} \vdash \S^2\mathbf{int}$.
It is then easy to see that, if $f(x_1, \ldots, x_n)$ and $g(y, y_1, \ldots, y_m)$ have been attributed types $1^p; \mathbf{int}; \ldots; \mathbf{int} \vdash \S^p\mathbf{int}$ and $1^q; \mathbf{int}; \ldots; \mathbf{int} \vdash \S^q\mathbf{int}$, then the function $g(f(x_1, \ldots, x_n), y_1, \ldots, y_m)$ can be given a similar type, namely $1^{p+q}; \mathbf{int}; \ldots; \mathbf{int} \vdash \S^{p+q}\mathbf{int}$. In other terms, all polynomials in which each variable occurs exactly once can be typed.

Weakening and contraction

In order to get all polynomials, we must be able to represent dummy dependencies, and repetition of variables, i.e. weakening and contraction for **int**.
Weakening can be defined as a sum with a function which is identically 0 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\alpha \multimap \alpha \vdash \alpha \multimap \alpha}{\S(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)}}{\S(\alpha \multimap \alpha); !(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)} \quad \dfrac{\dfrac{\alpha \vdash \alpha}{\vdash \alpha \multimap \alpha}}{[1] \vdash !(\alpha \multimap \alpha)}}{[1]; \mathbf{int}_\alpha; !(\alpha \multimap \alpha) \vdash \S(\alpha \multimap \alpha)}}{[1]; \mathbf{int}_\alpha \vdash \mathbf{int}_\alpha}$$

etc. .
Contraction is obtained by composition with a diagonal map, i.e. the function $diag(n) = < n, n >$. For this observe that the successor induces a map of type $\mathbf{int} \otimes \mathbf{int} \vdash \mathbf{int} \otimes \mathbf{int}$ (corresponding to the function $f(< n, n >) = < n+1, n+1 >$, which can be retyped $1; \mathbf{int} \otimes \mathbf{int} \vdash \mathbf{int} \otimes \mathbf{int}$, and 0 induces an object of type $\mathbf{int} \otimes \mathbf{int}$. By iteration, we get a function of type $[1]; \mathbf{int} \vdash \S\mathbf{int} \otimes \mathbf{int}$, corresponding to the map $f(n) = < n, n >$. Now, if compose with the diagonal map, it is clear that we can identify variables (in general the integer k will increase).
So all polynomials can be given a type $1^k; \mathbf{int}; \ldots; \mathbf{int} \vdash \S^k\mathbf{int}$, and we can even fix the value of k when the degree is known.
Similar weakening and contraction maps are available for the types $\mathbf{list^k}$ to be defined below, in particular for **bint**.

The predecessor

Last, but not least, we must type the predecessor, i.e. the function *pred* such that $pred(0) = 0, p(n+1) = n$. The predecessor gets the type $!1;\mathbf{int} \vdash \mathbf{int}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha\vdash\alpha}{\alpha^\dagger\vdash\alpha}\&_1\vdash \qquad \dfrac{\alpha\vdash\alpha \quad \alpha\vdash\alpha}{\alpha\vdash\alpha^\dagger}}{\alpha^\dagger\multimap\alpha^\dagger;\alpha\vdash\alpha}}{\alpha^\dagger\multimap\alpha^\dagger\vdash\alpha\multimap\alpha}}{\S(\alpha^\dagger\multimap\alpha^\dagger)\vdash\S(\alpha\multimap\alpha)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha\vdash\alpha}{1;\alpha\vdash\alpha}}{1,\alpha\multimap\alpha;\alpha^\dagger\vdash\alpha}\&_2\vdash \quad \dfrac{\dfrac{\alpha\vdash\alpha \quad \alpha\vdash\alpha}{\alpha\multimap\alpha;\alpha\vdash\alpha}}{1,\alpha\multimap\alpha;\alpha^\dagger\vdash\alpha}\&_2\vdash}{1,\alpha\multimap\alpha;\alpha^\dagger\vdash\alpha^\dagger}}{1,\alpha\multimap\alpha\vdash\alpha^\dagger\multimap\alpha^\dagger}}{[1];[\alpha\multimap\alpha]\vdash!(\alpha^\dagger\multimap\alpha^\dagger)}}{!1;!(\alpha\multimap\alpha)\vdash!(\alpha^\dagger\multimap\alpha^\dagger)}}{\dfrac{!1;\mathbf{int}_{\alpha^\dagger};!(\alpha\multimap\alpha)\vdash\S(\alpha\multimap\alpha)}{!1;\mathbf{int}_{\alpha^\dagger}\vdash\mathbf{int}_\alpha}}$$

etc., with $\alpha^\dagger = \alpha \& \alpha$. . The basic idea is to iterate, instead of f of type $\alpha\multimap\alpha$, the function f' in $\alpha^\dagger\vdash\alpha^\dagger$ such that $f'(x) =< x, f(x) >$ [9]. Eventually the first projection of the result is kept. Similar functions for k-lists can be defined.

2.4.3 Some data types

Familiar data types as well as the basic operations on them can be represented in **LLL**. We shall only need a type with n elements $\mathbf{bool^n}$ and the type of lists of tokens taken among m tokens, $\mathbf{list^m}$.

Booleans

The type $\mathbf{bool^k}$ is defined as $\forall\alpha.\ \S(\alpha \& \ldots \alpha \multimap \alpha)$; there are k occurrences of α to the left, and we agree on some (irrelevant) bracketing convention. In particular, $\mathbf{bool^2}$ (written more simply **bool**) is $\forall\alpha.\ \S(\alpha\&\alpha\multimap\alpha)$. Its erasure $\forall\alpha.\ \alpha\wedge\alpha\Rightarrow\alpha$ is one of the standard representations of booleans in $\mathcal{F}$. We can define proofs $\mathbf{b}_1,\ldots,\mathbf{b}_k$ of **bool**, by starting with one of the k canonical proofs of $\alpha \& \ldots \alpha \vdash \alpha$, and ending with $\multimap$, $\S$ and $\forall$-rules. Typically the boolean « false », $\mathbf{b}_2$ is :

$$\dfrac{\dfrac{\dfrac{\dfrac{\alpha\vdash\alpha}{\alpha\&\alpha\vdash\alpha}\&_2\vdash}{\vdash\alpha\&\alpha\multimap\alpha}}{\vdash\S(\alpha\&\alpha\multimap\alpha)}}{\vdash\mathbf{bool}}$$

whose erasure is the standard term $\Lambda\alpha\ \lambda x^{\alpha\&\alpha}\ \pi_2(x)$, which represents « false » in $\mathcal{F}$.

[9] f' is not quite linear in f, which is reflected by the [1] in the sequent $[1];\alpha\multimap\alpha\vdash\alpha^\dagger\multimap\alpha^\dagger$

If then else

We can give a type $1^1; \mathbf{bool^k}; A; \ldots; A \vdash A$ to the k-ary version of « if... then ... else ... », when A is a data type (it works when A is a boolean type or a type of lists). We give an example when A is **int** and $k = 2$, i.e. we try to "type" the function $f(true, n, m) = n, f(false, n, m) = m$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha\vdash\alpha \qquad \dfrac{\dfrac{\dfrac{\alpha\vdash\alpha \quad \alpha\vdash\alpha}{\alpha;\alpha\multimap\alpha\vdash\alpha}}{\alpha;\alpha\multimap\alpha;1\vdash\alpha} \quad \dfrac{\dfrac{\alpha\vdash\alpha \quad \alpha\vdash\alpha}{\alpha;\alpha\multimap\alpha\vdash\alpha}}{\alpha;1;\alpha\multimap\alpha\vdash\alpha}}{\alpha;1,\alpha\multimap\alpha;1,\alpha\multimap\alpha\vdash\alpha\,\&\,\alpha}}{\alpha\,\&\,\alpha;1,\alpha\multimap\alpha;1,\alpha\multimap\alpha;\alpha\vdash\alpha}}{\alpha\,\&\,\alpha;1,\alpha\multimap\alpha;1,\alpha\multimap\alpha\vdash\alpha\multimap\alpha}}{[1];\mathbf{bool}_\alpha;\S(\alpha\multimap\alpha);\S(\alpha\multimap\alpha)\vdash\S(\alpha\multimap\alpha)} \quad \dfrac{\alpha\multimap\alpha\vdash\alpha\multimap\alpha}{[\alpha\multimap\alpha]\vdash!(\alpha\multimap\alpha)}}{[1];\mathbf{bool}_\alpha;\mathbf{int}_\alpha;\S(\alpha\multimap\alpha);[\alpha\multimap\alpha]\vdash\S(\alpha\multimap\alpha)} \quad \dfrac{\alpha\multimap\alpha\vdash\alpha\multimap\alpha}{[\alpha\multimap\alpha]\vdash!(\alpha\multimap\alpha)}}{\dfrac{\dfrac{[1];\mathbf{bool}_\alpha;\mathbf{int}_\alpha;\mathbf{int}_\alpha;[\alpha\multimap\alpha]\vdash\S(\alpha\multimap\alpha)}{[1];\mathbf{bool}_\alpha;\mathbf{int}_\alpha;\mathbf{int}_\alpha;!(\alpha\multimap\alpha)\vdash\S(\alpha\multimap\alpha)}}{[1];\mathbf{bool}_\alpha;\mathbf{int}_\alpha;\mathbf{int}_\alpha\vdash\mathbf{int}\alpha}}$$

etc., with $\mathbf{bool}_\alpha = \S(\alpha\,\&\,\alpha \multimap \alpha)$. Weakening and contraction on $\mathbf{bool^k}$ can be defined in terms of generalized « if... then ... .else ... ».

Lists

We define $\mathbf{list^k}$ to be $\forall\alpha.\ (!(\alpha\multimap\alpha)\multimap(\ldots\multimap!(\alpha\multimap\alpha)\ldots))\multimap\S(\alpha\multimap\alpha)$, with k occurrences of $!(\alpha\multimap\alpha)$ to the left. So $\mathbf{list^1}$ is just **int**, and $\mathbf{list^2}$ is abbreviated into **bint** (binary integers).

We discuss the type **bint**, but our discussion applies to any type $\mathbf{list^k}$. First we observe that the empty list **emptylist** and more generally any finite list of digits 0 and 1 can be encoded by a proof of **bint**. This is more or less obvious, since $\mathbf{bint}^-$ is the usual $\mathcal{F}$-translation of binary lists. Concatenation of lists, $\frown$, can be represented by a proof of $\mathbf{bint};\mathbf{bint} \vdash \mathbf{bint}$, which is basically a binary version of $+$, and that we therefore skip. In particular the two successor functions $\cdot \frown 0$ and $\cdot \frown 1$ can both be given the type $\mathbf{bint}\vdash\mathbf{bint}$.

An important function is the « kind of list », of type $!1;\mathbf{bint}\vdash\mathbf{bool^3}$. On the empty list it yields the value $\mathbf{b}_1$, on a list ending with 0, it yields the value $\mathbf{b}_2$, and on a list ending with 1, it yields the value $\mathbf{b}_3$: let $\alpha^* = (\alpha\,\&\,\alpha)\,\&\,\alpha$, and introduce three proofs Π_f, Π_g and Π_h of $\alpha^*\vdash\alpha^*$ respectively corresponding to the functions $f(x) =<< \pi_1(\pi_1(x)), \pi_1(\pi_1(x)) >, \pi_1(\pi_1(x)) >$,
$g(x) =<< \pi_2(\pi_1(x)), \pi_2(\pi_1(x)) >, \pi_2(\pi_1(x)) >$,

$h(x) =<< \pi_2(x), \pi_2(x) >, \pi_2(x) >$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\alpha \vdash \alpha \qquad \begin{matrix}\vdots\,\Pi_f\\ \alpha^* \vdash \alpha^*.\end{matrix}}{\alpha^* \multimap \alpha \vdash \alpha^* \multimap \alpha}}{\S(\alpha^* \multimap \alpha^*) \vdash \S(\alpha^* \multimap \alpha)} \quad \dfrac{\begin{matrix}\vdots\,\Pi_h\\ \alpha^* \vdash \alpha^*.\end{matrix}}{\overline{[1] \vdash !(\alpha^* \multimap \alpha^*)}}}{[1]; !(\alpha^* \multimap \alpha^*) \multimap \S(\alpha^* \multimap \alpha^*) \vdash \S(\alpha^* \multimap \alpha)} \quad \dfrac{\begin{matrix}\vdots\,\Pi_g\\ \alpha^* \vdash \alpha^*.\end{matrix}}{\overline{[1] \vdash !(\alpha^* \multimap \alpha^*)}}}{[1]; \mathbf{bint}_{\alpha^*} \vdash \S(\alpha^* \multimap \alpha)}}{\overline{!1; \mathbf{bint} \vdash \S(\alpha^* \multimap \alpha)}}}{!1; \mathbf{bint} \vdash \mathbf{bool}}$$

with $\mathbf{bint}_\alpha =!(\alpha \multimap \alpha) \multimap (!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha))$.
Among the functions connected with $\mathbf{list^k}$ are all the functions $\mathbf{list^f}$, of type $\mathbf{list^k} \vdash \mathbf{list^{k'}}$, induced by a map f from $\{1, \ldots, k\}$ to $\{1, \ldots, k'\}$. They are easily defined, mainly by structural manipulations. Three important examples :

- The (unique) function $\mathbf{list^f}$ from **bint** to **int** identifies the two digits, and produces a tally integer : it will be used for the length of the input of a Turing machine ;
- When $k \geqslant 2$, the function from **bint** to $\mathbf{list^k}$ that identifies a binary integer with a k-list : it will be used for the input tape of a Turing machine ;
- When $k \geqslant 2$, the function from $\mathbf{list^k}$ to **bint** that replaces any digit distinct from $0, 1$ with 0 : it will be used for the output of a Turing machine.

2.4.4 Polytime functions

Turing machines

Consider a (deterministic) Turing machine using p symbols and with q states. The current configuration can be represented by three data :

- a list dealing with the leftmost part of the tape (up to the position of the head)
- a list dealing with the right part of the tape, in reverse order
- the current state

The type $\mathbf{Tur^{p,q}} = \mathbf{list^p} \otimes \mathbf{list^p} \otimes \mathbf{bool^q}$ can therefore be used to represent any configuration of the machine. The instructions of the machine depend on reading the last symbol of one list, (including testing whether or not a list is empty), and also depend on the current state. From what precedes, it is possible (by eventually adding new instructions so that the machine can never stop), to represent a Turing machine by a proof of $!1; \mathbf{Tur^{p,q}} \vdash \mathbf{Tur^{p,q}}$: just use successors, predecessors, « kind of list », and generalized « if ... then ... else... ».

Inputs and outputs

We assume that our inputs are binary integers, i.e. that the digits 0 and 1 belong to the p legal symbols of the tape. The input (initial configuration) can therefore be expressed by means of a map of type $\mathbf{bint} \vdash \mathbf{Tur}^{\mathbf{p},\mathbf{q}}$ which maps a binary list s into the 3-tuple $< s, \mathbf{emptylist}, \sigma >$, where σ is the initial state. When the expected runtime of the machine is over, we may also decide to read off the output, i.e. we need to represent the map $f(< s, t, \tau >) = s'$, where s' is obtained from s by replacing any symbol distinct from $0, 1$ by 0. Such a function is easily obtained by means of a function $\mathbf{list}^{\mathbf{f}}$ and of the weakening facilities on our data types.

Run of a Turing machine

Assume that we are given a time θ (represented by a tally integer of type $\mathbf{int}$), an initial input s of type $\mathbf{bint}$, and a Turing machine of type
$!1; \mathbf{Tur}^{\mathbf{p},\mathbf{q}} \vdash \mathbf{Tur}^{\mathbf{p},\mathbf{q}}$. Then running the machine for θ steps from the initial input, can be represented by means of an iteration. As a function of θ, s it may receive the type $1^2; \mathbf{int}; \S\mathbf{bint} \vdash \S\mathbf{Tur}^{\mathbf{p},\mathbf{q}}$ and therefore (using the coercion map $!1; \mathbf{bint} \vdash \S\mathbf{bint}$) also the type $1^2; \mathbf{int}; \mathbf{bint} \vdash \S\mathbf{Tur}^{\mathbf{p},\mathbf{q}}$). The result at time θ (if we stop the machine after θ steps) can be written as a function of θ, s of type $1^2; \mathbf{int}; \mathbf{bint} \vdash \S\mathbf{bint}$.

Polytime machines

A polytime machine is machine with a polynomial clock, which stops after $P(\sharp(s))$ steps, where $\sharp(s)$ is the size of the input, and P is a given polynomial ; when $P(\sharp(s))$ steps have been executed, then we print out the result. Now observe that P can be given a type $1^k; \mathbf{int} \vdash \S^k\mathbf{int}$, and using the (unique) map $\mathbf{list}^{\mathbf{f}}$ from $\mathbf{bint}$ into $\mathbf{int}$, the function $P'(s) = P(\sharp(s))$ can be given the type $1^k; \mathbf{bint} \vdash \S^k\mathbf{int}$. By composition with the runtime function, we get the type $1^{k+2}; \mathbf{bint}; \S^k\mathbf{bint} \vdash \S^{k+1}\mathbf{bint}$ to represent the function $\varphi(s, s')$ which is the result of the computation after $P'(\sharp(s))$ steps with the input s'. Using the contraction facility on $\mathbf{bint}$) we can make $s = s'$ and replace this type with $1^{k+3}; \mathbf{bint} \vdash \S^{k+2}\mathbf{bint}$. If we insist on having the same integer on both sides, we can, using the coercion of type $1^{k+3}; \S^{k+2}\mathbf{bint} \vdash \S^{k+3}\mathbf{bint}$, replace this type with $1^{k+3}; \mathbf{bint} \vdash \S^{k+3}\mathbf{bint}$.

2.4.5 The representation theorem

Theorem 1

> *Any polytime function from binary lists to binary lists can be represented in* **LLL** *as a proof of a formula* $1^k; \mathbf{bint} \vdash \S^k\mathbf{bint}$

Proof. — This is obvious from what precedes. The algorithm can be executed in $\mathcal{F}$, but also as a proof-net, in which case the output is a proof-net with conclusions $\S^k\mathbf{bint}; \perp^k$, and the $\perp^k$, which eventually comes from 0-ary $\perp$-links, can be ignored. □

2.4.6 The user's viewpoint

Let us admit that this works, without being especially friendly. Since this paper is concerned with showing that **LLL** is intrinsically polytime, this subsection was concerned with a rather marginal question : to show that it was strictly polytime, i.e. that any polytime function could be typed inside the system. So we didn't care much about the potential users of such a system. Surely this practical aspect should be developed, under the form of a typed λ-calculus, analogous to system $\mathcal{F}$. The best would be system of pure λ-calculus with typing declarations in **ILLL**.

But this is not the only possibility : a less conservative option would be to exploit the classical symmetries of **LLL**, which have a lot of interesting consequences (for instance, using the fact that $\alpha \multimap \alpha$ is isomorphic to $\alpha \multimap \alpha$, which induces an isomorphism between $\mathbf{bint}_{\alpha^\perp}$ and $\mathbf{bint}_\alpha$, we get the proof :

$$\cfrac{\cfrac{\mathbf{bint}_{\alpha^\perp} \vdash \mathbf{bint}_\alpha}{\mathbf{bint} \vdash \mathbf{bint}_\alpha}}{\mathbf{bint} \vdash \mathbf{bint}}$$

whose action is to reverse a list).

3 PROOF-NETS FOR LLL

Cut-elimination in sequent calculus is unmanageable -especially in presence of additive features- : too many permutations of rules occur, and the counting of these permutations blurs the actual complexity of the process. This is why we choose to use *proof-nets* to prove the main theorem of this paper. Our basic reference will be [2] where the proof-net technology is expounded. We shall therefore content ourselves with modifying the definitions of [2] so as to take care of the specificities of **LLL**. We adopt the definitions and conventions of this paper, in particular we shall very often speak of formulas to mean "occurrences of formulas". We shall ignore the additive constants $\top$ and 0 on the double grounds that they play little role and that they can be handled anyway by means of second-order definitions in case we badly insist to keep them. This will save a lot of inessential details.

3.1 Proof-nets with multiplicative/additive conclusions

We first liberalize the condition about the weights of conclusions in definition 3 of [2]. Let $\Gamma = [\Delta]; \mathbf{A_1}; \ldots; \mathbf{A_n}$ be a sequent. Then a proof-structure will be declared to have the conclusion Γ when its conclusions are the formulas (discharged or not) listed in Γ and furthermore, for each $\mathbf{A_i}$ the sum of the weights of the formulas of $\mathbf{A_i}$ is equal to 1. This is equivalent to saying that, after applying *ad hoc* $\oplus$-links to the formulas of $\mathbf{A_i}$, then we obtain a proof-structure in the sense of section 3 of [2].

We consider the following exponentials links :

- The ?-link, with n unordered premises, which are all occurrences of the same discharged formula $[A]$, and with conclusion $?A$;
- The !-box, which is a generalized axiom whose (unordered) conclusions are $[A_1], \ldots ; [A_n]; !B$. This link is called a box because in order to use it, one has to give a proof-net Θ whose conclusions are $A_1, \ldots, A_n ; B$; our conventions about proof-structures imply that n is nonzero. A pictural representation of a box is precisely a... box whose contents is Θ and below which the conclusions of the link are written ;
- The §-box, which is a generalized axiom whose (unordered) conclusions are $[A_1]; \ldots ; [A_n]; \S A_{n+1}; \ldots ; \S A_{n+m}$. This link is called a box because in order to use it, one has to give a proof-net Θ whose conclusion is a sequent $\Gamma ; \Delta$ without discharged formulas, and such that the formulas occurring in Γ are exactly $A_1, \ldots, A_n$, and Δ is $\S A_{n+1}; \ldots ; \S A_{n+m}$. A typical example is that of a proof-net with conclusions $A, B; C; D, E, F; G; H$, which can be used to form a §-box with conclusions $[A]; [B]; [C]; [D]; [E]; [F]; \S G; \S H$, but also a §-box with conclusions $[A]; [B]; \S C; [D]; [E]; [F]; \S G; \S H$.

Weights are subject to the usual conditions ; moreover

- A discharged formula is the conclusion of exactly one link, i.e. one box ;
- If L is a ?-link with premises $[A_1], \ldots, [A_n]$ (occurrences of the same discharged formula), then $w(L) \geqslant w([A_i])$ for $i = 1, \ldots, n$; remember that a default jump, i.e. a formula B such that $w(B) \geqslant w(L)$ must be provided with the link.

The condition for being a proof-net is defined in the obvious way : once a valuation ψ has been selected, one builds a graph whose vertices are those formulas A such that $\psi(w(A)) = 1$. The edges are selected as in [2] ; moreover

- For any ?-link, one draws an edge between the conclusion of the link and any premise of the link which a vertex of the graph, or with the default jump B (this is crucial in case no premise of the link is a vertex of the graph) ;
- For any box with conclusions $A_1, \ldots, A_n$, one draws an edge between A_1 and A_2, A_2 and A_3,... A_{n-1} and A_n. The choice of edges depends on an ordering of the conclusions of the box, but any other ordering would produce an equivalent graph.

Observe that since boxes are built from proof-nets, our condition indeed means that a proof-structure is a proof-net iff it is a proof-net when we consider its boxes as proper axioms, and if the contents of its boxes are in turn proof-nets, etc.

3.2 Sequentialization for LLL

We must first define what it means for a proof in sequent calculus to be a *sequentialization* of a proof-net. This is done without problem, following the lines of [2]. We only need to be careful about the structural maintenance : typically certain formulas of $\vdash \Gamma$ are not present in the proof-net, because they would receive the weight 0. This is the case inside blocks, and for discharged formulas. We can state the :

Theorem 2

Proof-nets are sequentializable, i.e. every proof-net is the sequentialization of at least one sequent calculus proof.

Proof. — By induction on the *depth*, i.e. the maximum nesting of boxes. If we assume that the inside of all boxes is sequentializable, since the rules for the formation of boxes are the same as the rules for ! and §, then we are left with the problem of sequentializing a usual proof-net with boxes, a question solved in [2], section 3. □

3.3 Cut-elimination for LLL

Since this proof is rather delicate, we suggest to first understand it in the case without additives. Hence there is notion of weight, the !-boxes have exactly two conclusions, all cuts are ready, and all exponential cuts are special. Moreover the notion of proof-net in this case is akin to the more familiar multiplicative case.

3.3.1 The size and depth of a proof-net

Definition 4

The size $\sharp(L)$ *of a link* L *is defined by :*

- *if* L *is an identity link,* $\sharp(L) = 2$;
- *if* L *is a cut-link,* $\sharp(L) = 0$;
- *if* L *is an exponential box constructed from a proof-net with conclusion* Γ, *then* $\sharp(L) = 1 + s$, *where* s *is the number of semi-columns in* Γ ;
- *otherwise* $\sharp(L) = 1$.

The size $\sharp(\Theta)$ *of a proof-net* Θ *is the sum of the sizes of the links occurring in it, including what (hereditarily) occurs inside the boxes.*

Definition 5

The depth $\partial(\Theta)$ *of a proof-net* Θ *is the maximum nesting number of boxes in* Θ. *The* depth *of a formula* A *(denoted* ∂A *or* $\partial A/\Theta$*) is the number of boxes containing it : typically, if* Θ *consists of a sole box* $\mathcal{B}$ *made from a proof-net* Θ', *then the conclusions of* $\mathcal{B}$ *have depth* 0, *whereas the depth of a formula* A *of* Θ' *is given by* $\partial A/\Theta = \partial A/\Theta' + 1$. *One similarly defines the notion of* depth *of a link : typically in the case just considered, the box gets the depth* 0, *whereas* $\partial L/\Theta = \partial L/\Theta' + 1$ *for all links* L *occurring inside* $\mathcal{B}$. *Finally we define the* partial size, *also called d-size,* $\sharp_d(\Theta)$ *to be the sum of the sizes of links of depth* d *in* Θ, *so that* $\sharp(\Theta) = \sharp_1(\Theta) + \ldots + \sharp_n(\Theta)$, *where* n *is the depth of* Θ.

These definitions have been chosen because of their relevance to cut-elimination. But what about the relevance of our size w.r.t. the actual size of a proof-net ?

- The size of a link is almost the number of its conclusions. In Θ, define a function f as follows : if A is not discharged, let$f(A)$ be any link with conclusion A, if $[A]$ is discharged, then it is the conclusion of a box, and A occurs (undischarged) inside the box, and we set $f([A]) = f(A)$. It is easy to see that L occurs in the range of f at most twice the size of L, hence the number of formulas in Θ is bounded by $2\sharp(L)$.

- Cut-links do not contribute to the size ; however if A is the premise of such a link, then A is the conclusion of another link, and it is easy to see that the number of cuts cannot exceed the size.

- The actual size of a net as a graph is therefore linear in the official size, Good News ! However we are not done since the net also involves the boolean weights. But, as observed in [2], these weight can be replaced with a structure of coherent space between the links and therefore the size of the missing structure is quadratic in the official size of the net.

- Finally the size does not take into account the actual sizes of formulas. Here vary little can be done, especially in presence of quantifiers. The most natural viewpoint is to see the formulas as comments, which are erased at runtime, in the same way that the actual execution of a typed λ-term is the execution of its erasure, i.e. of the underlying pure λ-term. In other terms, we work with a kind of *interaction net* à la Lafont, see [6].

This should be enough to convince one that the polynomial bounds obtained below actually induce a polytime algorithm. Concretely, as explained in [2], the substitutions occurring during the additive steps are delayed and those occurring during the quantifier steps are not performed (they can be stored in some auxiliary memory). If the final result should be without additives, then the additive substitutions can be done at the end, producing a cleansing of the graph (all weights become 0 or 1). If the final result is also free from any kind of existential quantifiers, then the formulas can be synthesized in an obvious way, and we have no use for our stack of substitutions.

3.3.2 Cut-elimination : the general pattern

We shall define a *lazy* cut-elimination which terminates in polytime. The result of the procedure (which is Church-Rosser) is cut-free only in certain cases, but this is enough for us.

Let us call a cut *exponential* when the cut-formulas begin with exponentials and both premises are conclusions of exponential links. For non-exponential ready links, the paper [2] defines a linear time cut-elimination procedure : each step of this *basic* procedure strictly shrinks the size of the proof-net (and this remains true with our specific measurement of size).The pattern is as follows :

- In a preliminary round we apply the basic procedure at depth 0, which induces a shrinking of the proof-net at depth 0, the other sizes staying the same ; then the real things begin

- In a first round we work at depths 0 and 1 ; at depth 1 only the basic procedure is allowed, whereas only certain exponential cuts are removed at depth 0. If the original partial sizes were $s_0, \ldots, s_d$, then the new sizes after completing the procedure, will not exceed $s_0, s_0 s_1, \ldots, s_0 s_d$ (and the depth does not increase).

- In a second round we apply a similar procedure at depths 1 and 2 ; this procedure fires no new reduction at depth 0, so that after completing this second round, our partial sizes will not exceed $s_0, s_0 s_1, s_0^2 s_1 s_2, \ldots, s_0^2 s_1 s_d$, and the depth still not increases.

- The d^{th} round occurs at depths $d-1$ and d. When it is completed, nothing more can be done (in the lazy case, we shall be cut-free). The depth of the proof-net is still at most d (it can diminish in the very unlikely situation of erasure of a deeply nested box), and the sizes are now at most $s_0, s_0 s_1, s_0^2 s_1 s_2, s_0^4 s_1^2 s_2 s_3, \ldots, s_0^{2^d} s_1^{2^{d-1}}, \ldots, s_{d-2}^2 s_{d-1} s_d$. The final size is therefore bounded by s^{2^d}.

It will be easy to see that s^{2^d} actually counts the number of steps, if s is the size and d is the depth : we are therefore polystep in s (when d is fixed, which corresponds to practice). Since the steps are not to big, the actual runtime is polynomial in the number of steps, and the complexity of cut-elimination, for a given depth d will therefore be polytime.

3.3.3 Elimination of exponential cuts

Definition 6

The actual weight *of a discharged formula* $[A]$ *is the weight of the conclusion* A *of the proof-net inside the box. An exponential cut is* special *if it is a ready and in case one of the premises of the cut is the conclusion of* a ?-*link, then this link is either* 0*-ary or one of its premises has actual weight* 1.

We now explain how to eliminate special cuts : this is the *special procedure*

- §-reduction : take a ready cut between $\S A$ and $\S A^\perp$, both A and $A^\perp$ are conclusions of §-boxes whose contents are proof-nets with respective conclusions $\Gamma; A$ and $A^\perp; \Delta$: in this case we first perform a cut on A between the two proof-nets, yielding a proof-net with conclusion $\Gamma; \Delta$, then we form a §-box with this proof-net.

- Weakening reduction : take a special cut between $!A$ and $?A^\perp$, where $?A^\perp$ is the conclusion of a 0-ary link : in this case we remove the box with conclusion $!A$. This involves the destruction of the conclusions $[B_i]$ of this box, but this only amounts to reducing the arity of some ?-links.

- Contraction reduction : take a special cut between $!A$ and $?A^{\perp}$, where $?A^{\perp}$ is the conclusion of a ?-link with a premise $[A_i{}^{\perp}]$ of actual weight 1. Then $[A_i{}^{\perp}]$ is in turn the conclusion of a box $\mathcal{B}$. $\mathcal{B}$ is made from a proof-net Ξ whose conclusions are $A_i{}^{\perp};\Delta$, whereas the box $\mathcal{A}$ with $!A$ among its conclusions is made from a proof-net Θ whose conclusions are $\Gamma;A$. By means of a cut between A and $A_i{}^{\perp}$, we can produce a new proof-net Π. Π can be used to produce a new box $\mathcal{C}$ whose conclusions are the same as those of $\mathcal{B}$, except that $[A_i{}^{\perp}]$ is replaced with $[\Gamma]$. In this case we replace $\mathcal{B}$ with $\mathcal{C}$. Observe that new occurrences of $[\Gamma]$ are created, hence the arity of some ?-link will increase.

What about the size during this procedure ? Let us assume that our special cut is of depth 0, and that our original sizes are $s_0, s_1, \ldots, s_d$;

- §-reduction : the size obviously decreases by 2, since three links (two boxes and a cut) counting for $1+n+1+m$ are replaced with two links (one box and a cut), counting for $1+(n-1)+(m-1)+1$. A new estimate for the partial sizes is $s_0-2, s_1, \ldots, s_d$;

- Weakening reduction : the size strictly shrinks, and $s_0-1, s_1, \ldots, s_d$ is a very pessimistic majorization of the size of the result ;

- Contraction reduction : at depth 0 the size stays the same, since $\mathcal{C}$ has the same size as $\mathcal{B}$ (this is because there is no semicolon in Γ, so that $\Gamma;\Delta$ has the same number of semi-colons as $A_i{}^{\perp};\Delta$). But otherwise it increases : more precisely, if the partial sizes of the proof-net Θ are $t_0, t_1, \ldots, t_{d-1}$, then the partial sizes of our new proof-net are exactly $s_0, s_1+t_0, s_2+t_1, \ldots, s_d+t_{d-1}$.

In the first round we systematically perform the basic procedure at depth 1 together with the special procedure at depth 0. The point of the basic procedure is that it induces changes of weights inside the boxes, and therefore some conclusions of the proof-nets inside a box receive a new weight 0, in which case some conclusion $[A]$ of the box disappears. This does not affect the size of the box (the number of semicolons stays the same) and since such a conclusion was the premise of some ?-link, this only induces a change of arity of the ?-link. Of course some conclusion of box may get the actual weight 1, which can fire a contraction reduction, etc. By the way no basic reduction at depth 0 can be fired during the first round, and this is why we may assume that they have been done during a preliminary round. Later on, in the second round, no basic reduction at depth 0 or 1 will occur etc.

3.3.4 Bounding the sizes

Bounding the size essentially amounts to considering the first round. We therefore assume that the basic procedure has been completed at depth 0. We also make a simplifying hypothesis, namely that no non-trivial weight remains at depth 0 : this will be the case when we normalize proofs of *lazy sequents*, see

below [10].
We introduce a *precedence* relation between discharged formulas : $[A] <_1 [B]$ when $[B]$ is conclusion of a !-box $\mathcal{B}$ and the other conclusion of the box $!A^{\perp}$ is the premise of an exponential cut whose other premise $?A$ is the conclusion of a ?-link, with $[A]$ among its premises. By the correctness criterion, the transitive closure $<$ of precedence is a partial order. We can therefore consider the forest $\mathcal{F}$ of finite sequences $([A_0], \ldots, [A_n])$ of discharged formulas, such that $[A_0]$ is minimal w.r.t. $<$ and $([A_0] <_1 \ldots <_1 [A_n])$. A *discharged block* is a set of discharged formulas $[\mathbf{A}]$ which occur among the conclusion of some exponential box of depth 0, made from a proof-net with conclusion Γ, and such that $[\mathbf{A}]$ is a block of Γ. A *coherent subforest* in $\mathcal{F}$ is a subforest μ of $\mathcal{F}$ such that whenever two sequences $[S], [A], [S']$ and $[S], [B], [S'']$ belong to μ, then either $[A]$ and $[B]$ are the same or they belong to distinct discharged blocks. Given μ, we can define the *multiplicators* $\mu(\mathcal{B}$ for any box $\mathcal{B}$ with discharged conclusions to be the number of sequences in μ such that the last element of the sequence is a conclusion of B ; if $\mathcal{B}$ is a §-box whose conclusions are all of the form $\S A$, let $\mu(\mathcal{B}) = 1$. The *potential sizes* of Θ are defined as follows : for each depth $i \neq 0$, we can write $s_i = \sum s_i^{\mathcal{B}}$, where $\mathcal{B}$ varies through boxes of depth 0 ($s_i^{\mathcal{B}}$ is just the contribution of the proof-net inside $\mathcal{B}$ to i^{th} size). We define $S_0^{\mu} = s_0, S_1^{\mu} = \sum \mu(\mathcal{B}) s_1^{\mathcal{B}}, \ldots, S_d^{\mu} = \sum \mu(\mathcal{B}) s_d^{\mathcal{B}}$.

Proposition 1

> *Assume that Θ reduces to Π during the first round. Then the potential sizes of Π are not greater than the potential sizes of Θ.*

Proof. — We already know that the size does not increase at depth 0 ; let us check the property at any other depth, typically depth 1, and in the only problematic case, namely the contraction reduction. We start with boxes $\mathcal{A}$ and $\mathcal{B}$ in Θ to produce a box $\mathcal{C}$ which replaces $\mathcal{B}$. If a_1, b_1 are the respective contributions of $\mathcal{A}$ and $\mathcal{B}$ to the 1-size of Θ, then then contribute as $\mu_{\Theta}(\mathcal{A})a_1 + \mu_{\Theta}(\mathcal{B})b_1$ to the potential 1-size of Θ, whereas in Π the boxes $\mathcal{A}$ and $\mathcal{B}$ contribute to the potential size as $\nu_{\Pi}(\mathcal{A})a_1 + \nu_{\Pi}(\mathcal{C})(a_1 + b_1)$. But since $\mathcal{C}$ is obtained by merging $\mathcal{B}$ with a copy of $\mathcal{A}$, it is easy to construct, given ν a μ such that $\mu(\mathcal{B}) = \nu(\mathcal{C})$ and $\mu(\mathcal{A}) = \nu(\mathcal{C}) + \nu(\mathcal{A})$. This proves the claim. □
By the way observe that any maximal μ will yield $\mu(\mathcal{A}) \geqslant 1$, hence the potential sizes easily exceeds the sizes ; on the other hand observe that coherent subforests are not too big, since they cannot branch at all : this is due to the peculiarities of the ! boxes. Moreover, thanks to acyclicity, the same discharged formula cannot occur twice in the same branch : in other terms $\mu(\mathcal{A})$ cannot exceed the number of roots of μ [11], which is bounded by the number of discharged blocks, and this number is in turn bounded by s_0. This is why the first round yields the bounds $s_0, s_0 s_1, \ldots, s_0 s_d$.

[10] If non-trivial weights appear at depth 0, we can apply the constructions of this subsection to the part of the proof-net which is of size 1

[11] This is the only point where **ELL** diverges from **LLL** : in **ELL** coherent subforests do branch !

3.3.5 Bounding the runtime

We show below that the runtime is of degree 3 in the size, which will yield polytime complexity of degree 2^{d+2} for our algorithm. It suffices to compute the complexity of the first round :

- The number s_0 dominates both the number of steps of the preliminary round and the number of special steps which are not contraction reductions ; the number $s_0 s_1$ dominates the number of basic steps in the first round. The number of contraction reductions performed during the first round is smaller than the maximum size of a coherent subforest of $\mathcal{F}$, and is therefore less than b_0^2, where b_0 is the number of discharged blocks of depth 0, which is turn is bounded by s_0^2. The number of steps during the first round is therefore easily bounded by $(s_0 + s_1)^2$;

- However, the number of steps is not the runtime : some steps, typically contraction reductions involve a duplication of the structure, which means that each step can cost at most the *actual size* of the proof-net. We already observed that the actual size is quadratic in the size (which is bounded by $s_0 s$) hence we arrive at a total of $(s_0 s)^2 s^2$ for the first round.

Without being very cautious, we can bound the total runtime by something like $3s^{2^d}$, which is enough for our purpose.
This does not mean that the algorithm cannot be improved. The decomposition in rounds is rather artificial etc. But we are not looking for efficient implementation, just for a proof-system which is intrinsically polytime, and that's it.

3.3.6 Lazy sequents

A formula is said to be *lazy* when it contains neither the symbol & nor higher order existential quantification. A sequent is said to be *lazy* when all the formulas occurring in it are lazy.

Proposition 2

Let Θ be a proof-net without non-exponential ready cut of depth 0, *and assume that Θ has a ready cut at depth* 0. *Then one of the conclusions of Θ is a non-lazy formula of weight* 1.

PROOF. — since eigenweights can only be used at a given depth, some eigenweight is used at depth 0, and we can look for a &-link L such that the empire of its conclusion is maximal w.r.t. any valuation. Then the downmost conclusion below this link cannot be the premise of a cut (in which case we can show, as in [2], that the cut would be ready), hence it must be a conclusion, and its weight is bigger that the weight of L, so it is equal to 1. □
As a corollary, after the preliminary round, a proof-net whose conclusion is lazy has no non-trivial weight at depth 0.

Proposition 3
After the first round, the proof of a lazy sequent has no cut of depth 0.

Proof. — assume that the first round is completed, and consider the forest $\mathcal{F}$; if there is still a cut of depth 0, then there is a sequence $([A_0], [A_1])$ in $\mathcal{F}$, and $[A_0]$ is the conclusion of a box $\mathcal{B}$. Since a conclusion of $\mathcal{B}$ is the hereditary premise of an exponential cut and the contraction reduction does not apply, then this conclusion must have a non-trivial weight. Now the proof-net Π, which is in $\mathcal{B}$ has a conclusion with a non-trivial weight, and since the basic procedure has been completed for Π, there is a conclusion C of Π which is non-lazy and of weight 1. This conclusion yields a conclusion of $\mathcal{B}$ and :

- either the conclusion is a formula $\S C$; since this formula is non-lazy, it must be the premise of a cut... But the §-reduction would apply, a contradiction ;
- or this conclusion is a formula $!C$, which must also be the premise of a cut. In this case, observe that $C <_1 A_0$, a contradiction ;
- or this conclusion is the premise $[C]$ of a ?-link, which must in turn be the premise of a cut ; in this case $[C]$ is of actual weight 1, and the contraction elimination does apply, a contradiction.

Therefore $\mathcal{F}$ is trivial and Θ is cut-free at depth 0. □

Theorem 3
Cut-elimination converges to a (unique) normal form for proofs of lazy sequents ; furthermore, for bounded depth, the runtime is polynomial in the size of the net.

Proof. — more or less obvious from what precedes. □
Observe that application of a function of type $1^k; \mathbf{bint} \vdash \S^k\mathbf{bint}$ to an argument falls into this case : a (cut-free) argument is of depth 1, hence the global depth is the depth of the proof representing the function. Observe that, unlike other approaches, like [7, 5], there is still a complexity bound for arbitrary functionals, something like n^{2^n}, since the size clearly exceeds the depth by 2.

A APPENDIX

A.1 Naive Set-Theory and LLL

We have so far only considered second order propositional **LLL**. But this is not the only possibility :

- We can consider first-order **LLL**, which is straightforward.
- We can also consider **LLL** with first and second order quantifications ; this system would be a natural candidate for a light second order arithmetic. By the way a light first order arithmetic could easily be extracted, but one would

have to think twice in front of the difficulties inherent to equality, especially in terms of proof-nets (e.g. certain formulas like $0 \neq 1$ will be equivalent to $\top$, hence the case of $\top$ has first to be fixed) ;

- We can also consider quantifications of any order...

- And last but not least, we can consider Naive Set Theory, which encompasses all kinds of quantification.

In fact, Naive Set Theory has been the starting point of **LLL** : I was looking for a system in which the complexity could be expressed independently of the complexity of the cut-formulas. In particular it would also work for naive set-theory, since there is a well-known (non-terminating, for obvious reasons) cut-elimination procedure for it ; by the way, it had been observed long ago by Grishin [4] that, in the absence of contraction, cut-elimination works [12]. So I decided to translate Russell's paradox into linear logic with exponentials. Using fixpoint facilities (see below) one can produce a new constant A, which has the rules (unary links in terms of proof-nets) : from $?A$ deduce $A^{\perp}$, from $!A^{\perp}$ deduce A.

- There is a first possibility for deriving a paradox (here a proof-net with no conclusion), which is based on dereliction : the proof-net has depth 1, but the process of cut-elimination does not converge at depth 0, since the normalization of a cut with dereliction between $?A$ and $!A^{\perp}$ involves an "opening" of the box with conclusion $!A^{\perp}$: the contents of this box is "poured" into depth 0, so that the size s_0 no longer shrinks ;

- There is another possibility which does not use dereliction, but the principle $??A \multimap ?A$; in this case, the first round is easily completed, but the handling of the exponential cuts involves the creation of a deeper box, i.e. the size increases.

This is why we restricted to rules whose normalization involves no change of depth.

A.1.1 Expressive power of LLLs

Light Naive Set-Theory **LLLs** is defined exactly as **LLL**, but for the quantifiers and the terms :

Definition 7

Terms (T) and formulas (F) are defined as follows

$T = x, y, z, \ldots \ \{x \mid F\}$

$F = T \in U, T \notin U, 1, \perp, 0, \top, !F, \S F, ?F, F \otimes F, F \parr F, F \& F, F \oplus F, \forall x F, \exists x F$

[12] This is not very helpful, since the system without exponentials is awfully inexpressive in terms of computational power

Negation is defined as expected ; in particular, $(T \in U)^{\perp} = T \notin U$ and $(T \notin U)^{\perp} = T \in U$.
The logical rules are modified as follows :

$$\frac{\vdash \Gamma; A}{\vdash \Gamma; \forall x\, A} \ \textit{(for all : } \alpha \textit{ is not free in } \Gamma) \qquad \frac{\vdash \Gamma; A[T/x]}{\vdash \Gamma; \exists x\, A} \ \textit{(there is)}$$

$$\frac{\vdash \Gamma; A[T/x]}{\vdash \Gamma; T \in \{x \mid A\}} \ (\in) \qquad \frac{\vdash \Gamma; A[T/x]^{\perp}}{\vdash \Gamma; T \notin \{x \mid A\}} \ (\notin)$$

The representation in terms of proof-nets is straightforward : the $\in$-rules induce two unary links, one with premise $A[T/x]$ and conclusion $T \in \{x \mid A\}$, the other with premise $(A[T/x])^{\perp}$ and conclusion $T \notin \{x \mid A\}$.

A.1.2 Equality

Definition 8

The Leibniz equality $t = u$ is defined by $\forall x(t \in x \multimap u \in x)$; $t \neq u$ is short for $t = u \multimap 0$, a strong form of negation.

Exercise. — Prove the following sequents :

- ▶ $t = u; A[t/x] \vdash A[u/x])$
- ▶ $t = u \vdash u = t$
- ▶ $t = u \vdash (u = v \multimap t = v)$
- ▶ $t = u \vdash 1$
- ▶ $t = u \vdash t = u \otimes t = u$

Definition 9

The singleton $\{t\}$ is defined as $\{x \mid x = t\}$; the pair $\{t, u\}$ is defined as $\{x \mid x = t \oplus x = u\}$; the ordered pair $< t, u >$ is defined as $\{\{t\}, \{t, u\}\}$.

Exercise. — Prove the following sequents

- ▶ $\{t\} = \{t'\} \vdash t = t'$
- ▶ $\{t, u\} = \{t', u'\} \vdash (t = t' \oplus t = u') \otimes (u = t' \oplus u = u')$
- ▶ $\{t, u\} = \{t', u'\} \vdash (t = t' \otimes u = u') \oplus (t = u' \otimes u = t') \oplus t = u$
- ▶ $\{t, u\} = \{t', u'\} \vdash (t = t' \otimes u = u') \oplus (t = u' \otimes u = t')$
- ▶ $\{t\} = \{t', u'\} \vdash t = t' \otimes t = u'$
- ▶ $< t, u > = < t', u' > \vdash t = t' \otimes u = u'$

Exercise. — Prove the formula $\{x \mid 0\} \neq \{t\}$; conclude that we can find terms $t_0, \ldots, t_n, \ldots$ such that $t_i \neq t_j$ is provable for $i \neq j$.
As a consequence of the exercise, it is possible to represent certain features of the usual equalitarian predicate calculus :

- ▶ we can represent a n-ary function letter f by assigning to it a specific term t_i coming from the previous exercise ; $ft_1 \ldots t_n$ will be represented as $< a_i, t_1, \ldots, t_n >$, using a $n+1$-ary pairing function. It follows from the previous exercises that usual equality axioms are satisfied together with $ft_1 \ldots t_n \neq gu_1 \ldots u_m$ (when f, g are distinct) and $ft_1 \ldots t_n = gu_1 \ldots u_n \multimap t_1 = u_1 \otimes \ldots \otimes t_n = u_n$
- ▶ we can also represent predicates by means of fixed variables (generic constants) and by means of the pairing function : $pt_1 \ldots t_n$ becomes $(t_1, \ldots, t_n) \in x$, where x is a variable assigned to p
- ▶ as a consequence, we have access to a representation of binary strings : for this we only need a constant ϵ and two unary successors S_0, S_1. Equality axioms, as well as inequalities $S_0 t \neq S_1 u, S_0 t \neq \epsilon, S_1 t \neq \epsilon$ are provable, as well as $S_i t = S_i u \multimap t = u$ for $i = 1, 2$.

A.1.3 Fixpoints

In order to formulate the fixpoint property, we introduce the following notation : the substitution of an abstraction term $\lambda x_1 \ldots x_n.B$ for a n-ary predicate symbol P in the formula A consists in replacing any atom $Pt_1 \ldots t_n$ of A by $B[t_1/x_1, \ldots, t_n/x_n]$.

Proposition 4

Let A be a formula in the language of **LLLs** *augmented by means of a n-ary predicate P, and let $x_1, \ldots, x_n$ be variables, so that we can write our formula $A[P, x_1, \ldots, x_n]$; then there is a formula B (depending on $x_1, \ldots, x_n$) such that the equivalence (i.e. both linear implications) between $A[\lambda x_1 \ldots x_n.B[x_1, \ldots, x_n]$ and B is provable.*

PROOF. — This is a straightforward imitation of Russell's paradox (already used in the fixpoint theorem of λ-calculus). For instance, let us assume that $n = 1$; then we can form $t := \{z \mid \exists x \exists y.z =< x, y > \otimes A[\lambda w. < w, y > \in y, x]\}$. Then $< x, t > \in t$ is provably equivalent with $A[\lambda w. < w, t > \in t, x]$ and we are done. □

As a consequence we get the possibility of defining various partial recursive functions. Typically, take for instance the exponential function (defined on binary strings, i.e. in number base 2) then we can get a two variable formula B such that $B[s, t]$ expresses that $t = 2^s$.

This is enough to convince one that **LLLs** bears all the features of a light arithmetic. In particular all numerical functions implicit in proofs made in this system will be polytime computable.

A.2 Elementary Linear Logic

Elementary Linear Logic arise as the alternative solution to the complexity problem at stake. It syntax does not contain § (or rather does not need it). The rule for ! is liberalized into :

$$\frac{\vdash B_1 \mid \ldots \mid B_n ; A}{\vdash [B_1]; \ldots ; [B_n]; !A} \quad (\textit{of course})$$

where the symbols $B_1, \ldots, B_n$ are separated by commas or semicolons.
As a consequence, the sequent $!A; !(A \multimap B) \vdash !B$ becomes provable (equivalently $!A; !B \vdash !(A \otimes B)$ is provable). Integers can now be represented by the type $\forall\alpha.\ !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$: the tally integers can be given this type, which was not the case for **LLL**. The representation results of **LLL** persist (replace § by ! everywhere). We can also get rid of the irritating markers 1^k in the representation theorem, since the rule for ! is now valid with an empty context. But new functions arise, namely exponentials. This is due the fact that multiplication can now be given the type **int**; **int** $\vdash$ **int**. If we feed the first argument with the integer $\overline{2}$, we can type duplication with **int** $\vdash$ **int**, and as soon as duplication can be given a type $A \vdash A$, then we can iterate it, yielding a representation of the exponential function. The exponential can therefore be typed with **int** $\vdash$!**int**, and towers of exponentials with the type **int** $\vdash !^k$**int**. The same holds for other data types, and therefore we conclude that all elementary functions (i.e. functions whose runtime is bounded by a tower of exponentials) can be typed in **ELL**.
Is this optimal ? The proof of normalization still holds [13], but for the fact that coherent subforests are not so simple, since they may branch. The multiplication factor involved in the first round is no longer s_0 but depends exponentially on s_0, something like $s_0^{s_0}$. Completing the process will therefore cost a tower of exponentials, the height of the tower depending on the depth of the proof-net. Hence normalization is elementary in the size of the input, when the depth is given. This is analogous to the familiar bounds for predicate calculus/simply typed λ-calculus, but here the height of the tower does not depend on the cut-formula, but on more hidden parameter, the depth.
It is also possible to build a naive set-theory **ELLs**. Its expressive power is considerably bigger than before, since the exponential function plays a decisive role in mathematics. This induces a strange system which can both formalize a bunch of mathematics, and which admits definition by fixpoint. Such a system seems to be the optimal candidate for formalization of AI.

A.3 Questions

Semantics

What is the natural semantics for **LLL** ? We can of course take the usual semantics of linear logic, e.g. coherent spaces, but we shall be embarrassed to explain why certain principles are wrong. This question is presumably the deepest connected with our new system : for the first time polynomial time appears as the result of the free application of logical principles which are in no means contrived

[13] Even with a non-lazy procedure : normalizing non-ready cuts increases the size by an exponential factor.

to achieve this goal. A semantics of **LLL** would therefore be a general semantics of polytime. This might be very rewarding : remember that polytime has been characterized in many ways, but always through *presentations* « A function is polytime iff it can be obtained by means of... », and nobody knows how to deal with a presentation. On the other hand a semantic characterization would insist on something like preservation properties etc. that a mathematician can more easily reason about.

The connective « § »

This strange connective has been introduced to compensate two things, namely the want of dereliction, but also the failure of the principle [V] : $!A \otimes !B \vdash !(A \otimes B)$, which is essential in the representation of data types. Surely $\S A \otimes \S B \vdash \S(A \otimes B)$ holds and $\S(A \oplus B) \vdash \S A \otimes \S B)$ fails, but there are principles (typically the self-duality of the connective) that have been added on the sole grounds of their simplifying character. Later investigations (in particular semantical ones) could help to clarify this question. In a similar way, the fact that !1 is not provable is backed by good taste (!1 looks like the 0-ary case of [V]), but by no deep intuition.

Completeness

In some sense **LLL** and **ELL** are complete, since the complexity bounds are here once for all. This is even more conspicuous with their naive set-theoretic extensions : what could be more powerful than unrestricted comprehension ? In some sense the theorems, the algorithms coming from these systems should be absolute. Is it possible to make sense of this informal remark ?

Execution

In our systems, the runtime is known in advance, depending only on the depth and size. We could seek an untyped calculus, with a notion of depth, and for each depth d a function $r_d(.)$ with the following property : after $r_d(\sharp(t))$ steps, then we reach either a normal form or a deadlock. There should be two solutions, corresponding to polytime and elementarity.

BIBLIOGRAPHY

[1] J.-Y. Girard. **Linear logic.** *Theoretical Computer Science*, 50:1–102, 1987.

[2] J.-Y. Girard. **Proof-nets : the parallel syntax for proof-theory**. In Ursini and Agliano, editors, *Logic and Algebra*, New York, 1995. *Marcel Dekker.*

[3] J.-Y. Girard, A. Scedrov, and P.J. Scott. **Bounded Linear Logic: A Modular Approach to Polynomial Time Computability**. *Theoretical Computer Science*, 97:1–66, 1992.

[4] V.N. Grishin. **Predicate and set-theoretic calculi based on logics without contractions**. *Math. USSR Izvestiya*, 18:41–59, 1982.

[5] G.G. Hillebrand, P.C. Kanellakis, and H.G. Mairson. **Database query languages embedded in the typed lambda calculus**. In *Proc. 8-th Annual IEEE Symposium on Logic in Computer Science, Montreal*, pages 332–343, June 1993.

[6] Y. Lafont. **From proof-nets to interaction nets**. In Girard, Lafont, and Regnier, editors, *Advances in Linear Logic. Cambridge University Press*, 1995.

[7] D. Leivant. **A foundational delineation of poly-time**. *Information and Computation*, 110:391–420, 1994. (Special issue of selected papers from LICS'91, edited by G. Kahn.).

[8] D. Leivant and J.-Y. Marion. **Lambda calculus characterizations of poly-time**. *Fundamenta Informaticae*, 19:167–184, 1993. (Special Issue: Lambda Calculus and Type Theory, edited by J. Tiuryn.).

Intrinsic Theories and Computational Complexity

Daniel Leivant

Computer Science Department, Indiana University

Abstract. We introduce a new proof theoretic approach to computational complexity. With each free algebra $\mathbb{A}$ we associate a first order "intrinsic theory for $\mathbb{A}$", $\mathbf{IT}(\mathbb{A})$, with no initial functions other than the constructors of $\mathbb{A}$, and no axioms for them other than the generative and inductive axioms, which delineate $\mathbb{A}$. The case most relevant to traditional proof theory is $\mathbb{A} = \mathbb{N}$ (the unary natural numbers), and the case most relevant to computer science is $\mathbb{A} = \mathbb{W} = \{0,1\}^*$. An algorithm is *provable* if it provably maps inputs in $\mathbb{A}$ to values in $\mathbb{A}$, and a function is provable if it has a provable algorithm. We show that the provable functions of $\mathbf{IT}(\mathbb{N})$ are exactly the provably recursive functions of Peano Arithmetic.

We further show that function provability is equivalent to computational complexity for the following pairs theory/complexity-class: (1) A ramified variant $\mathbf{RT}(\mathbb{A})$ of $\mathbf{IT}(\mathbb{A})$ and elementary functions. (2) $\mathbf{RT}(\mathbb{W})$ with quantifier-free induction and poly-time; (3) $\mathbf{RT}(\mathbb{N})$ with quantifier-free induction and linear space (on register machines).

Intrinsic theories combine lean axiomatics with expressive flexibility, since they permit explicit (uncoded) reference to arbitrary computable functions. Thus, the characterizations above provide user-friendly formalisms for feasible mathematics, in which non-feasible algorithms can be mentioned freely. Moreover, natural deduction calculi for these formalisms correspond directly, via formula-as-type homomorphisms, to applicative programs.

1 Intrinsic theories

1.1 Intrinsic theories for free algebras

Let $\mathbb{A}$ be a free algebra, generated from constructors $\mathbf{c}_1 \ldots \mathbf{c}_k$, where $arity(\mathbf{c}_i) = r_i \geq 0$.[2] Let $arity(\mathbb{A}) =_{\mathrm{df}} \max_i r_i$. A free algebra of arity 1 is a *word algebra*. For example, the word algebra $\mathbb{N}$ with constructors $\mathbf{0}$ and $\mathbf{s}$ (of arity 0 and 1 respectively) is isomorphic to the natural numbers, and the word algebra $\mathbb{W}$, with

[*] Work partially supported by NSF grant CCR-9309824

[2] That is, $\mathbf{c}_i$ are function identifiers, and $\mathbb{A}$ consists of the closed terms generated from them.

constructors ϵ, $\mathbf{0}$ and $\mathbf{1}$ (of arity 0,1, and 1 respectively) is essentially $\{0,1\}^*$; e.g. $\mathbf{0}(\mathbf{1}(\mathbf{1}(\epsilon)))$ can be identified with 011.

The *full intrinsic theory for* $\mathbb{A}$, $\mathbf{IT}^=(\mathbb{A})$, is a theory in first order logic with equality, defined as follows. The vocabulary consists of a unary relation identifier A and the constructors of $\mathbb{A}$.[3] For a list $\mathbf{t} = \mathbf{t}_1 \ldots \mathbf{t}_r$ of terms, we let $A(\mathbf{t})$ abbreviate the conjunction $A(\mathbf{t}_1) \wedge \cdots \wedge A(\mathbf{t}_r)$. The axioms of $\mathbf{IT}(\mathbb{A})$ are:

- *Generative axioms:* $\forall u_1 \ldots u_{r_i} \quad A(\mathbf{u}) \rightarrow A(\mathbf{c}_i(\mathbf{u})) \quad (i = 1 \ldots k)$.

- *Limitative axioms:* All instances of $\mathbb{A}$-Induction,[4]

$$\forall x \,(\, A(x) \rightarrow Cl_A[\lambda u.\varphi] \rightarrow \varphi[x]_u \,)$$

$$\text{where } \; Cl_{\mathrm{A}}[\lambda u.\varphi] \;\equiv_{\mathrm{df}}\; \textstyle\bigwedge_{i=1\ldots k} Cl_{\mathbf{c}_i}[\lambda u.\varphi]$$

$$Cl_{\mathbf{c}_i}[\lambda u.\varphi] \;\equiv_{\mathrm{df}}\; \forall v_1 \ldots v_{r_i} \;\; (\, \varphi[v_1] \wedge \cdots \wedge \varphi[v_{r_i}] \rightarrow \varphi[\mathbf{c}_i(\mathbf{v})] \,)$$

The axioms for equality are $\forall x.x = x$, and all formulas of the form $\forall x, y.\, (x = y \,\wedge\, \varphi[x] \rightarrow \varphi[y])$, where φ is atomic. Finally, let $\mathbf{IT}^=_o(\mathbb{A})$ denote the fragment of $\mathbf{IT}^=(\mathbb{A})$ with induction restricted to existential formulas.[5]

1.2 Provable functions

We use as computation model Herbrand-Gödel style equational programs, as defined e.g. in [Lei94b], and we refer to the notion of coherent program defined there. This model lends itself to direct rendition in intrinsic theories, without coding or auxiliary concepts or notations, and without invoking formalisms that allow non-denoting terms (see §6.2). If P is an equational program, we write $\bar{P}$ for the conjunction of the universal closures of all equations in P. The following observation seems to be part of the folklore (see [Lei94b] for a proof).

THEOREM **1.1** *Let* $(P, \mathbf{f})$ *be a coherent equational program over* $\mathbb{A}$. *The function* f *over* $\mathbb{A}$ *computed by* $(P, \mathbf{f})$ *is total iff* $\bar{P} \rightarrow \; A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$ *is true in every model of* $\bar{P}$ *in which the denotation of* A *equals the set of denotations of* $\mathbb{A}$*-terms.*

This equivalence motivates the following definition. Let T be a theory whose vocabulary contains that of $\mathbf{IT}^=(\mathbb{A})$. We say that a function f over a free algebra $\mathbb{A}$ is *provable in* T iff it is computed by some equational program $(P, \mathbf{f})$ over $\mathbb{A}$, such that $T \vdash \bar{P} \rightarrow A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$.[6]

[3] We make an analogous typographical convention for specific algebras; for instance, W is used for the algebra $\mathbb{W}$, and N for $\mathbb{N}$.

[4] We let un-parenthesized implications associate to the right. Also, we write $E[t]_u$ for the result of substituting t for all free occurrences of variable u in the expression E. We omit the subscript when in no fear of confusion.

[5] That is, formulas whose canonical prenex form is purely existential.

[6] We use boldface $\mathbf{f}$ as formal identifier for the function f, and similarly for other characters.

Of course, an intrinsic theory can be assigned, more generally, to any data system, i.e. to a sorted structure whose universes are defined by simultaneous closure conditions (see e.g. [Lei90a]). We do not refer here to such generalizations.

1.3 Classically and constructively provable functions

It is well known that the same functions over N are provably recursive in first order arithmetic regardless of whether the underlying logic is classical, constructive (intuitionistic), or minimal.[7] The proof of the analogous statement for provable functions is trivial:

THEOREM **1.2** *If a function over* $\mathbb{A}$ *is provable in* $\mathbf{IT}^{=}(\mathbb{A})$ *using classical logic, then it is provable already using minimal logic.*

Proof. Assume that $\bar{P} \rightarrow A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$ is provable in $\mathbf{IT}^{=}(\mathbb{A})$ based on classical logic. By standard double-negation translations (see e.g. [Lei85]), this implies that $\bar{P} \rightarrow A(\mathbf{x}) \rightarrow \neg\neg A(\mathbf{f}(\mathbf{x}))$ is provable using minimal logic, i.e. $\bar{P} \rightarrow A(\mathbf{x}) \rightarrow ((A(\mathbf{f}(\mathbf{x})) \rightarrow \bot) \rightarrow \bot)$ is provable. Since minimal logic has no rule for $\bot$, the latter formula remains provable in minimal logic after (correct) substitution of any formula for $\bot$. Substituting $A(\mathbf{f}(\mathbf{x}))$ yields $\bar{P} \rightarrow A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$. □

1.4 Equality-free intrinsic theories

Let P be an equational program over $\mathbb{A}$. Say that terms $\mathbf{t}$ and $\mathbf{t}'$ are *P-equivalent* if there is an equation $\mathbf{q} = \mathbf{s}$ in P, and a substitution σ, such that $\mathbf{t}'$ is obtained from $\mathbf{t}$ by replacing an occurrence of $\sigma\mathbf{q}$ by $\sigma\mathbf{s}$, or vice versa. Let $\mathbf{IT}(\mathbb{A})$ be the fragment of $\mathbf{IT}^{=}(\mathbb{A})$ without equality, and let $\mathbf{IT}(\mathbb{A}, P)$ be $\mathbf{IT}(\mathbb{A})$ with, as additional axioms, all formulas $A(\mathbf{t}) \rightarrow A(\mathbf{t}')$, where $\mathbf{t}'$ is P-equivalent to $\mathbf{t}$. $\mathbf{IT}_o(\mathbb{A})$ and $\mathbf{IT}_o(\mathbb{A}, P)$ are the corresponding theories with induction restricted to existential formulas. We say that $(P, \mathbf{f})$ is *provable in* $\mathbf{IT}(\mathbb{A})$ if $\mathbf{IT}(\mathbb{A}, P) \vdash A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$.

PROPOSITION **1.3** *An equational program* $(P, \mathbf{f})$ *is provable in* $\mathbf{IT}^{=}(\mathbb{A})$ *iff it is provable in* $\mathbf{IT}(\mathbb{A})$. *Similarly for* $\mathbf{IT}_o^{=}(\mathbb{A})$ *and* $\mathbf{IT}_o(\mathbb{A})$.

Proof. If $(P, \mathbf{f})$ is not provable in $\mathbf{IT}(\mathbb{A})$ then, by completeness, $\mathcal{S} \not\models A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$ for some model $\mathcal{S}$ of $\mathbf{IT}(\mathbb{A}, P)$. Let $\mathcal{S}^{=}$ be $\mathcal{S}$ augmented with the interpretation of equality as $\{\langle a, b\rangle \mid a = b$ *is derived from P using names for all elements of* $\mathcal{S}\}$. Then $\mathcal{S}^{=}$ is a model of $\mathbf{IT}^{=}(\mathbb{A}) + P$ but not of $A(\mathbf{x}) \rightarrow A(\mathbf{f}(\mathbf{x}))$, so $(P, \mathbf{f})$ is not provable in $\mathbf{IT}^{=}(\mathbb{A})$. The proposition's forward implication holds also for constructive and for minimal logic.[8] By Theorem 1.2. The backward implication is trivial. □

We use a natural deduction calculus for $\mathbf{IT}(\mathbb{A}, P)$, where

[7] The simplest proof of this fact is due to H. Friedman [Fri78]. Minimal logic, also called *positive logic*, is the fragment of constructive logic with no rules for negation or falsehood.

[8] A proof theoretic argument, not given here, yields low complexity bounds on the increase of proof size.

- The generative axioms of $\mathbf{IT}(\mathbb{A})$ are formulated as inference rules: from the r_i premises $A(\mathbf{t}_1),\dots A(\mathbf{t}_{r_i})$ (where $\mathbf{t}_1\dots,\mathbf{t}_{r_i}$ are terms) infer $A(\mathbf{c}_i(\mathbf{t}_1\dots\mathbf{t}_{r_i}))$.
- The $\mathbb{A}$-induction principle is formulated as an inference rule: From the $k+1$ premises $A(\mathbf{t})$ and $Cl_{\mathbf{c}_i}[\lambda u.\varphi]$ $(i=1\dots k)$ infer $\varphi[\mathrm{t}]_u$.[9]
- An inference rule $[P]$: from a formula $A(\mathbf{t})$ infer $A(\mathbf{t}')$, provided $\mathbf{t}'$ is P-equivalent to $\mathbf{t}$. By Proposition 1.2 we may assume, without loss of generality, that implication is the only propositional connective used, and $\forall$ the only quantifier.[10] Alternatively, one may easily expand the development below to apply to the remaining logical constants.

1.5 The provable functions of Peano Arithmetic

Recall that a function $f:\mathrm{N}\to\mathrm{N}$ is *provably recursive* in Peano Arithmetic **PA** if $f(x)=\mu y.\varphi$ where φ is a formula of **PA** with all quantifiers bounded, with x,y as the only free variables, and such that $\mathbf{PA}\vdash\forall x\exists y.\ \varphi$. In this section we show that every provably recursive function of **PA** is a provable function of $\mathbf{IT}(\mathrm{N})$.

Let **FA** be **PA** with function identifiers for all primitive recursive functions, and with the defining equations for them.[11] Let $\mathbf{FA}^-$ be **FA** without Peano's third and fourth axioms. The following is proved in [Lei91].

LEMMA **1.4** *If f is a provably recursive function of* **PA** *then there is a primitive recursive function g such that $f(x)=\mu y.\ (g(x,y)=0)$ and* $\mathbf{FA}^-\vdash\forall x\exists y\,\mathrm{g}(x,y)=\mathbf{0}$.

A straightforward induction on primitive recursive function definitions establishes the following

LEMMA **1.5** *Every primitive recursive function is provable in* $\mathbf{IT}(\mathrm{N})$.

LEMMA **1.6** *If a formula φ is provable in $\mathbf{FA}^-$, by a proof theat refers to functions $f_1\dots f_m$, then φ^N (that is, φ with all quantifiers relativized to N) is provable in $\mathbf{IT}^=(\mathrm{N})$, from the defining equations for $f_1\dots f_m$.*

Proof. Straightforward induction on proofs. Lemma 1.5 is used for the quantifier rules. □

THEOREM **1.7** *Every provably recursive function of* **PA** *is provable in* $\mathbf{IT}^=(\mathrm{N})$.

[9] Note that this is an *inference rule* in the sense of natural deduction, and *not* the Induction Proof Rule (cf. e.g. [Par77]); its restriction to a class Φ of formulas is therefore equivalent to the Induction Schema restricted to Φ.

[10] In this context "existential formula" should be read "quantifier free formula."

[11] For all applications it suffices to take Kalmar-elementary functions.

Proof. Suppose that f is a provably recursive function of **PA**. By Lemma 1.4

$$\mathbf{FA}^- \vdash \forall x \exists y\ \mathbf{g}(x,y) = \mathbf{0} \tag{1}$$

where g is as above. By Lemma 1.6 it follows that

$$\mathbf{IT}^=(\mathrm{N}),\ P_0 \vdash \forall x^N \exists y^N\ \mathbf{g}(x,y) = \mathbf{0} \tag{2}$$

where P_0 consists of the defining equations for the functions used in the proof for (1).

Without loss of generality, we may assume that[12]

$$\forall x \exists ! y\, \mathbf{g}(x,y) = \mathbf{0} \tag{3}$$

Let P be P_0 augmented by the equations

$$\mathbf{h}(\mathbf{0},y) = y \quad \text{and} \quad \mathbf{f}(x) = \mathbf{h}(\mathbf{g}(x,y),y) \tag{4}$$

(where **h** and **f** are fresh). Then P is coherent by (3), and f is computed by $(P,\mathbf{f})$, i.e. $f(n) = m$ iff $\mathbf{f}(\bar{n}) = \bar{m}$ is derivable.[13] (Note that $\mathbf{h}(\mathbf{0},\bar{m}) = \bar{m}$ is an instance of P, and $\mathbf{g}(\bar{n},\bar{m}) = \mathbf{0}$ is derivable from P_0.) By (2) and (4) it follows that

$$\mathbf{IT}^=(\mathrm{N}),\ P \vdash N(x) \rightarrow N(\mathbf{f}(x)).$$

□

2 Provability and applicative programs

2.1 Recurrence over $\mathbb{A}$

It is well known that natural deductions of suitable calculi can be viewed as λ-terms.[14] In [Lei83, Lei90a] we modified this mapping to a homomorphism, which yields directly an applicative program for a function f from a derivation for the provability of f in various formalisms. This section is an expanded revision of [Lei90a, §5].

Let $\mathbf{1\lambda}$ be the typed lambda calculus. We let $\rightarrow$ associate to the right, and we write $\rho_1, \ldots, \rho_r \rightarrow \tau$ for $\rho_1 \rightarrow \cdots \rightarrow \rho_r \rightarrow \tau$; if all ρ_i's are the same type ρ, we write $\rho^r \rightarrow \tau$ for the above. Let $\mathbf{Rec}(\mathbb{A})$ be $\mathbf{1\lambda}$ augmented with the following constants: (1) for each constructor $\mathbf{c}_i$ of $\mathbb{A}$, a constant c_i of type $\sigma_i[o]$, where $\sigma_i[\rho] =_{\mathrm{df}} \rho^{r_i} \rightarrow \rho$ (recall that $r_i = \mathit{arity}(\mathbf{c}_i)$); (2) for each type τ, a constant $\mathbf{R}^{\mathbb{A}}_\tau = \mathbf{R}_\tau$ of type $o, \sigma_1[\tau], \ldots, \sigma_k[\tau] \rightarrow \tau$. The reduction rules of $\mathbf{Rec}(\mathbb{A})$ are: the β-rule, and for each type τ the rule of *recurrence in type* τ:

$$\mathbf{R}_\tau(c_i a_1 \ldots a_{r_i}) M_1 \cdots M_k \ \Rightarrow\ M_i A_1 \cdots A_{r_i} \qquad \text{where } A_j =_{\mathrm{df}} \mathbf{R}_\tau a_j M_1 \cdots M_k$$

[12] Otherwise replace g by $g'(x,y) =_{\mathrm{df}}$ **if** $\forall z < y.\, g(x,z) \neq 0$ **then** $g(x,y)$ **else** $1+g(x,y)$.

[13] $\bar{n} =_{\mathrm{df}} \mathbf{s}^{[n]}\mathbf{0} =$ the n'th numeral.

[14] This is the Schönfinkel-Howard isomorphism, also known as the Curry-Howard formula-as-type analogy [Sch24, How80].

We write $\mathbf{Rec}_o(\mathbb{A})$ for the fragment of $\mathbf{Rec}(\mathbb{A})$ with $\mathbf{R}_\tau$ used only for $\tau = o$.

A term is *normal* if no subterm can be reduced. The familiar Tait-Prawitz method [Pra71] is easily applicable to $\mathbf{Rec}(\mathbb{A})$, yielding

LEMMA **2.1** *Every reduction sequence in* $\mathbf{Rec}(\mathbb{A})$ *terminates.*

Note that each element of $\mathbb{A}$ is represented in $\mathbf{Rec}(\mathbb{A})$ by itself (modulo currying), with type o, and that every closed normal terms of type o is an element of $\mathbb{A}$. Thus, every expression of type $o^r \rightarrow o$ represents an r-ary function over $\mathbb{A}$.

Clearly, the constant $\mathbf{R}_\tau$ denotes the operation of *iteration* in type τ, that is the function $\langle g_1 \dots g_k \rangle \mapsto f$, where f is defined by $f(\mathbf{c}_i(a_1 \dots a_{r_i})) = g_i(f(a_1) \dots f(a_{r_i}))$. The following observation is therefore trivial.

PROPOSITION **2.2** *Every function over* N *represented in* $\mathbf{Rec}(\mathrm{N})$ *is generated by iteration in finite type (and explicit definitions).*

2.2 Proofs as applicative programs

The natural deduction calculus for $\mathbf{IT}(\mathbb{A}, P)$ (where P is any coherent program) can be mapped homomorphically to $\mathbf{Rec}(\mathbb{A})$, as follows. Define inductively a mapping κ from formulas to types by: $\kappa(A(\mathbf{t})) =_{\mathrm{df}} o$; $\kappa(\psi \rightarrow \chi) =_{\mathrm{df}} (\kappa\psi \rightarrow \kappa\chi)$; $\kappa(\forall x.\psi) =_{\mathrm{df}} \kappa\psi$. Define further a mapping from derivations of $\mathbf{IT}(\mathbb{A}, P)$ to terms of $\mathbf{Rec}(\mathbb{A})$, which without danger of ambiguity we also denote by κ. If Π is a derivation from labeled open assumptions $\psi_1^{j_1}, \dots, \psi_q^{j_q}$ to conclusion φ, then $\kappa\Pi$ will be a term of type $\kappa\varphi$, with free variables $x_{j_1} \dots x_{j_q}$, of types $\kappa\psi_1 \dots \kappa\psi_q$, respectively.[15] The term $\kappa\Pi$ is defined by recurrence on Π, as follows, using the convention that if Π is a derivation, then $\Pi_1, \Pi_2, \dots$ are Π's immediate sub-derivations, in that order.

1. If Π is a labeled open assumption ψ^j, then $\kappa\Pi = x_j^{\kappa\psi}$ (the j-th variable of type $\kappa\psi$).
2. If Π derives $\psi \rightarrow \varphi$ by implication introduction, closing labeled assumption ψ^j, then $\kappa\Pi = \lambda x_j^{\kappa\psi}.\kappa\Pi_1$.
3. If Π derives φ by implication elimination, from $\psi \rightarrow \varphi$ and ψ, then $\kappa\Pi = (\kappa\Pi_1)(\kappa\Pi_2)$.
4. If Π derives φ by universal introduction, universal elimination, or $[P]$, then $\kappa\Pi = \kappa\Pi_1$.
5. If Π derives $A(\mathbf{c}_i(\mathbf{t}_1 \dots \mathbf{t}_{r_i}))$ by the generative rule for $\mathbf{c}_i$, then $\kappa\Pi = c_i(\kappa\Pi_1) \cdots (\kappa\Pi_{r_i})$.
6. If Π derives $\varphi[\mathbf{t}]_u$ by $\mathbb{A}$-induction from $A(\mathbf{t})$ and $Cl_{\mathbf{c}_i}[\lambda u.\varphi]$ $(i = 1 \dots k)$, then $\kappa\Pi = \mathbf{R}_{\kappa\varphi}(\kappa\Pi_1)(\kappa\Pi_2) \cdots (\kappa\Pi_{k+1})$.

[15] We stipulate a concrete syntax for natural deductions, where those open assumptions ψ that are closed jointly at some inference are labeled by a common natural number, not used as a label elsewhere in the proof.

2.3 Provability equals definition by recurrence

The following three lemmas are straightforward.

LEMMA **2.3** *For every $a \in \mathbb{A}$ there is a normal deduction Θ_a of* **IT**($\mathbb{A}$), *deriving $A(a)$, such that $\kappa\Theta_a = a$.*

LEMMA **2.4** *If $\mathbf{d} = \kappa\Pi$ is a closed normal term and Π derives in* **IT**$(\mathbb{A}, P)$ *a formula of the form $A(\mathbf{t})$, then $\mathbf{d} \in \mathbb{A}$ and $P \vdash \mathbf{t} = \mathbf{d}$.*

LEMMA **2.5** *Suppose Π is a deduction deriving φ from assumptions $\psi_1 \ldots \psi_q$. If $\kappa\Pi$ reduces to E in* **Rec**($\mathbb{A}$), *then $E = \kappa\Pi'$ for some deduction Π', also deriving φ from $\psi_1 \ldots \psi_q$.*

THEOREM **2.6** *Suppose that the equational program $(P, \mathbf{f})$ computes a function f over $\mathbb{A}$. If Π is a deduction in* **IT**$(\mathbb{A}, P)$ *deriving $A(\mathbf{x}) \to A(\mathbf{f}(\mathbf{x}))$, then $\kappa\Pi$ represents f in* **Rec**($\mathbb{A}$).

Proof. Without loss of generality, let f be unary. Given $a \in \mathbb{A}$ let Π_a be the result of substituting a for free occurrences of x in Π. We have $\kappa\Pi_a = \kappa\Pi$ trivially. Combining Π_a and Θ_a by implication elimination yields a proof Σ_a of $A(\mathbf{f}(a))$, with $\kappa\Sigma_a = (\kappa\Pi_a)(\kappa\Theta_a)$, which by Lemma 2.3 is $= (\kappa\Pi)(a)$. By Lemma 2.1 $\kappa\Sigma_a$ reduces to a closed normal term, which by Lemma 2.5 is $\kappa\Sigma'_a$ for some proof Σ'_a of $A(\mathbf{f}a)$. By Lemma 2.4 $P \models \kappa\Sigma'_a = f(a)$, so $\kappa\Pi$ represents f. □

THEOREM **2.7** *A function f over $\mathbb{A}$ is provable in* **IT**($\mathbb{A}$) *iff it is definable in* **Rec**($\mathbb{A}$). *Similarly, f is provable in* $\mathbf{IT}_o(\mathbb{A})$ *iff it is definable in* $\mathbf{Rec}_o(\mathbb{A})$.

Proof. The forward implication follows from Theorem 2.6. The proof of the backward implication is straightforward (and not used in the sequel). □

THEOREM **2.8** [Göd58] *The provably recursive functions of* **PA** *are precisely the functions definable by iteration in all finite types.*

Proof. We prove the forward implication. If f is provably recursive in **PA**, then it is provable in $\mathbf{IT}^{=}$(N), by Theorem 1.7. From Proposition 1.3 it follows that f is provable in **IT**(N), and therefore is definable in **Rec**(N), by Theorem 2.7. Thus f is definable by recurrence in finite types. □

3 Ramified intrinsic theories

3.1 Ramified algebras

In this section we introduce ramified variants of intrinsic theories, and characterize in terms of ramified provability the Kalmar-elementary functions, the poly-time functions, and the linear-space functions. The conceptual rationale for ramification is discussed in §6.4 below.

Given a free algebra $\mathbb{A}$, we define the *ramified intrinsic theory for* $\mathbb{A}$, $\mathbf{RT}(\mathbb{A})$, based on first order logic without equality, and using the logical constants $\wedge$, $\rightarrow$, $\forall$ and (for classical and constructive logic) $\perp$. The vocabulary consists of the constructors of $\mathbb{A}$ and a countable list of unary relation identifiers, A_0, $A_1 \ldots$. The mathematical principles are formulated as inference rules, like for $\mathbf{IT}(\mathbb{A})$:

- *Generative rules:* for each $\ell \geq 0$, from $A_\ell(\mathbf{t}_1), \ldots, A_\ell(\mathbf{t}_{r_i})$ infer $A_\ell(\mathbf{c}_i(\mathbf{t}))$ $(i = 1 \ldots k)$.
- *Ramified* $\mathbb{A}$*-induction:* For each $\ell \geq 0$ and formula φ with no A_j with $j \geq \ell$, infer from the $k{+}1$ premises $A_\ell(\mathbf{t})$ and $Cl_{\mathbf{c}_i}[\lambda u.\varphi]$ $(i = 1 \ldots k)$ the formula $\varphi[\mathbf{t}]$.
- $\mathbb{A}$*-selection:* Infer from the $k{+}1$ premises $A_0(\mathbf{t})$ and $Cl^o_{\mathbf{c}_i}[\lambda u.\varphi]$ $(i = 1 \ldots k)$ the formula $\varphi[\mathbf{t}]$, where

$$Cl^o_{\mathbf{c}_i}[\varphi] \equiv_{\mathrm{df}} \forall v_1 \ldots v_{r_i} \;\; A_0(\mathbf{v}) \rightarrow \varphi[\mathbf{c}_i(\mathbf{v})].$$

Note that the un-ramified form of the Selection schema is proved by $\mathbb{A}$-Induction for $\lambda u.\, A(u) \wedge \varphi[u]$. However, the ramified Selection schema is not a special case of ramified $\mathbb{A}$-Induction, because φ may refer to A_ℓ for any ℓ. We write $\mathbf{RT}_o(\mathbb{A})$ for $\mathbf{RT}(\mathbb{A})$ with ramified induction restricted to formulas without $\rightarrow$ or $\forall$.[16]

3.2 Provable functions of ramified theories

Let T be a theory whose vocabulary contains that of $\mathbf{RT}(\mathbb{A})$. We say that a function f over a free algebra $\mathbb{A}$ is *provable in T* iff it is computed by some equational program $(P, \mathbf{f})$ over $\mathbb{A}$, such that, for some j,

$$T \vdash \bar{P} \rightarrow A_j(\mathbf{x}) \rightarrow A_0(\mathbf{f}(\mathbf{x})).$$

A trivial ramified induction establishes the following:

Lemma **3.1** *If $m \geq \ell \geq 0$ then* $\mathbf{RT}(\mathbb{A}) \vdash A_m(x) \rightarrow A_\ell(x)$.

From this it follows that $(P, \mathbf{f})$ is provable in $\mathbf{RT}(\mathbb{A})$ iff $\mathbf{RT}(\mathbb{A}) \vdash \bar{P} \rightarrow A_{j_1}(x_1) \rightarrow \cdots \rightarrow A_{j_r}(x_r) \rightarrow A_i(\mathbf{f}(\mathbf{x}))$ for some $j_1 \ldots j_r, \ell$.

[16] If $\exists$ and $\vee$ are present, they are permitted; thus induction is restricted to positive existential formulas.

PROPOSITION **3.2** *The provable functions of* $\mathbf{RT}(\mathbb{A})$ *and of* $\mathbf{RT}_o(\mathbb{A})$ *are closed under composition.*

Proof. Consider, without loss of generality, the composition $f(x) = g(h_1(x), h_2(x))$, where g, h_1 and h_2 are provable, that is $\mathbf{RT}(\mathbb{A})$ proves $\bar{P} \rightarrow A_j(x,y) \rightarrow A_0(\mathbf{g}(x,y))$ and $\bar{P} \rightarrow A_{\ell_q}(x) \rightarrow A_0(\mathbf{h}_q(x))$ $(q = 1,2)$ for some j, ℓ_1, ℓ_2, where P is a program defining f via g and h as above. Incrementing by j all the subscripts of A in the proofs of the latter formulas yields proofs of $\bar{P} \rightarrow A_{\ell_q+j}(x) \rightarrow A_j(\mathbf{h}_q(x))$ $(q = 1,2)$. Let $p = \max(\ell_1, \ell_2) + j$. Then, by Lemma 3.1, $\bar{P} \rightarrow A_p(x) \rightarrow A_j(\mathbf{h}_q(x))$ $(q = 1,2)$, and so $\bar{P} \rightarrow A_p(x) \rightarrow A_0(\mathbf{f}(x))$. □

We say that a program $(P, \mathbf{f})$ is *flatly provable* if $T \vdash \bar{P} \rightarrow A_0(\mathbf{x}) \rightarrow A_0(\mathbf{f}(\mathbf{x}))$. Analogously to Theorem 1.2 we have:

THEOREM **3.3** *If a function over* $\mathbb{A}$ *is provable (flatly provable) in* $\mathbf{RT}(\mathbb{A})$ *using classical logic, then it is provable (respectively, flatly provable) already using minimal logic.*

3.3 Examples of provable functions

Let us start with two flatly provable functions. Consider a free algebra $\mathbb{A}$ as above, with $r = \mathit{arity}(\mathbb{A})$. The *destructor functions* for $\mathbb{A}$ are the r functions defined by

$$\underline{\mathit{dstr}}_j(\mathbf{c}_i(a_1 \dots a_{r_i})) = \ \textbf{if } 1 \leq j \leq r_i \textbf{ then } a_i \textbf{ else } \mathbf{c}_i(a_1 \dots a_{r_i}) \qquad j = 1 \dots r.$$

The *case function* for $\mathbb{A}$ is the $(k+1)$-ary function $\underline{\mathit{case}} = \underline{\mathit{case}}_{\mathbb{A}}$, defined by $\underline{\mathit{case}}(\mathbf{c}_i(\mathbf{a}), x_1 \dots x_k) = x_i$.

LEMMA **3.4** *The destructor functions and the case function are flatly provable in* $\mathbf{RT}_o(\mathbb{A})$.

Proof. Let $\psi[u] \equiv_{\mathrm{df}} A_0(\underline{\mathit{dstr}}_j(u))$. For each $\mathbf{c}_i$ we have $A_0(v_1 \dots v_{r_i}) \rightarrow \psi[\mathbf{c}_i(\mathbf{v})]$, by the definition of $\underline{\mathit{dstr}}_j$. Thus $A_0(x) \rightarrow \psi[x]$ by Selection, so $\underline{\mathit{dstr}}_j$ is flatly provable. The proof for $\underline{\mathit{case}}$ is similar. □

Two simple examples of provable (but not flatly provable) functions are addition and multiplication. We phrase them as generic functions over word algebras $\mathbb{A}$:[17]

$$\begin{array}{lll} \oplus(\mathbf{c}_i, x) = x & \otimes(\mathbf{c}_i, x) = \mathbf{c}_i & \text{if } r_i = 0 \\ \oplus(\mathbf{c}_i(a), x) = \mathbf{c}_i(\oplus(a,x)) & \otimes(\mathbf{c}_i(a), x) = \oplus(x, \otimes(a,x))) & \text{if } r_i = 1 \end{array}$$

In the sequel we use $\oplus$ and $\otimes$ in infix whenever convenient. We are mostly interested in the fact that $\underline{\mathit{lngth}}(a \otimes b) = \underline{\mathit{lngth}}(a) \cdot \underline{\mathit{lngth}}(b)$.

[17] A yet more generic multiplicative function could be defined by $\odot(\mathbf{c}_i, x_1 \dots x_r) = x_i$, $\odot(\mathbf{c}_i(a), x_1 \dots x_r) = \oplus(x_i, \odot(a, \mathbf{x}))$.

LEMMA **3.5** *Let $\mathbb{A}$ be a word algebra. The functions $\oplus$ and $\otimes$ are provable in* $\mathbf{RT}_o(\mathbb{A})$. *In fact, for all $\ell \geq 0$ the formulas $A_{\ell+1}(y) \wedge A_\ell(x) \rightarrow A_\ell(y \oplus x)$ and $A_{\ell+1}(z) \wedge A_{\ell+1}(y) \rightarrow A_\ell(z \oplus y)$ are provable in* $\mathbf{RT}_o(\mathbb{A})$.

Proof. Let P be the defining equations above for $\oplus$ and $\otimes$. Let $\varphi[u] \;\; \equiv_{\rm df} A_\ell(u \oplus x)$. Under the assumption $A_\ell(x)$, the equations P imply $Cl_{c_i}[\lambda u.\varphi]$. So by ramified induction

$$A_{\ell+1}(y) \wedge A_\ell(x) \rightarrow A_\ell(y \oplus x) \tag{5}$$

Now let $\psi[v] \;\; \equiv_{\rm df} \;\; A_\ell(v \otimes y)$. Then, under the assumption $A_{\ell+1}(y)$ and $\bar{P}$ we have, by (5), $Cl_{c_i}[\lambda v.\psi]$. So by ramified induction on ψ we obtain

$$A_{\ell+1}(y) \wedge A_{\ell+1}(z) \rightarrow A_\ell(z \otimes y).$$

□

From Lemmas 3.5 and 3.2 we conclude

LEMMA **3.6** *All polynomial functions over $\mathbb{A}$ (i.e. compositions of $\oplus$ and $\otimes$) are provable in* $\mathbf{RT}_o(\mathbb{A})$.

Using induction for universal formulas we can further expand the collection of provable functions:

LEMMA **3.7** *Numeric exponentiation is provable in* $\mathbf{RT}(\mathbb{N})$.

Proof. Let P be the program with principal identifier $\underline{exp}$, and consisting of the three equations $\mathsf{e}(x, \mathbf{0}) = \mathsf{s}x$, $\mathsf{e}(x, \mathsf{s}y) = \mathsf{e}(\mathsf{e}(x,y),y)$, and $\underline{exp}(v) = \mathsf{e}(\mathbf{0}, v)$. Then, by induction on y, $\mathsf{e}(x,y) = x + 2^y$, so P computes the exponential function $\lambda n.2^n$.

Let $\varphi[u] \;\; \equiv_{\rm df} \; \forall v \; N_0(v) \rightarrow N_0(\mathsf{e}(v,u))$. We show $\varphi[u] \rightarrow \varphi[\mathsf{s}u]$. Assume $\varphi[u]$, and consider w. From $\varphi[u]$ we get $N_0(w) \rightarrow N_0(\mathsf{e}(w,u))$ as well as $N_0(\mathsf{e}(w,u)) \rightarrow N_0(\mathsf{e}(\mathsf{e}(w,u),u)))$. Combining these implications and using the defining equations for e, we get $N_0(w) \rightarrow N_0(\mathsf{e}(w, \mathsf{s}u))$, from which $\varphi[\mathsf{s}u]$ follows by $\forall$-introduction. Thus $\varphi[u] \rightarrow \varphi[\mathsf{s}u]$. Also, $\varphi[\mathbf{0}]$ is immediate from the generative axioms for N_0. It follows by induction that $N_1(x)$ implies $\varphi[x]$, in particular $N_0(\mathbf{0}) \rightarrow N_0(\mathsf{e}(\mathbf{0},x))$, and therefore $N_0(\mathsf{e}(\mathbf{0},x))$, i.e. $N_0(\underline{exp}(x))$. Thus $N_1(x) \rightarrow N_0(\underline{exp}(x))$. □

Define the functions 2_k $(k \geq 0)$ by $2_0(x) =_{\rm df} x$, and $2_{k+1}(x) =_{\rm df} 2^{2_k(x)}$. From Lemmas 3.2 and 3.7 we conclude

PROPOSITION **3.8** *All functions 2_k are provable in* $\mathbf{RT}(\mathbb{N})$. □

4 Ramified provability from complexity classes

4.1 Flat provability of machine transition functions

We relate provability to computation in a natural generic model of register machines (RMs) over free algebras, described in [Lei94c], to which we refer the reader for detail. A RM M consists of a set $S = \{s_1 \dots s_\ell\}$ of *states*, (s_1 and s_ℓ are the initial and terminal state, respectively), a list $\Pi = \pi_1 \dots \pi_m$ of *registers*, and a finite set of commands, each being a constructor command, a destructor command, or a branching command.[18] We posit some code $\#s_i \in \mathbb{A}$ for each state s_i.

LEMMA **4.1** *Given a deterministic RM M, there are $m+1$ $(m+1)$-ary functions $\tau_0, \tau_1, \dots, \tau_m$ that are flatly provable in $\mathbf{RT}_o(\mathbb{A})$, and such that configuration $[s, u_1, \dots, u_m]$ has an M-transition to configuration $[s', u'_1, \dots, u'_m]$ iff $\tau_i(\#s, u_1, \dots, u_m) = u'_i$ for $i = 1 \dots m$, and $\tau_0(\#s, u_1, \dots, u_m) = \#s'$. Moreover, if M has no transition for state s, then $\tau_i(u_0 \dots u_m) = u_i$ $(i = 0 \dots m)$.*

Proof. Each u'_j $(j = 1 \dots m)$ is either u_j, or is obtained by applying a constructor or a destructor of $\mathbb{A}$ to some u_i's. Moreover, the appropriate case for the u'_j's and for the state-code argument $\#s'$ can all be determined from the input using the function $\underline{case}$. All these functions are flatly provable in $\mathbf{RT}_o(\mathbb{A})$ by Lemma 3.4, so $\tau_0 \dots \tau_m$ are also flatly provable in $\mathbf{RT}_o(\mathbb{A})$. □

4.2 Provability of functions

THEOREM **4.2** *Let $\mathbb{A}$ be a word algebra. If a function f over $\mathbb{A}$ is computable on a RM over $\mathbb{A}$ in polynomial time then f is provable in $\mathbf{RT}_o(\mathbb{A})$.*

Proof. We give the proof for $\mathbb{A} = \mathbb{W}$; the general case is similar. Assume there is an m such that $f(w)$ is computed in $\leq |w|^m$ steps for all $w \in \mathbb{W}$. Write τ for the vector of functions $\langle \tau_0 \dots \tau_m \rangle$, and let $\alpha = \langle \#s_1, w, \epsilon, \dots \epsilon \rangle$ be the initial configuration of M. By Lemma 3.6 there is a program $(P_m, \mathbf{T})$ for $\lambda n.n^m$ such that

$$\mathbf{RT}_o(\mathbb{W}) \vdash \bar{P}_m \to W_\ell(x) \to W_1(\mathbf{T}(x)) \qquad (6)$$

for a suitable $\ell \geq 1$. Let P consist of the defining equations of the functions τ_j, the equations in P_m, and the following:

$$\underline{cfg}_j(\epsilon, x) = \text{ the } j\text{'th entry of } \alpha$$

$$\underline{cfg}_j(\mathrm{c}t, x) = \tau_j(\underline{cfg}_0(t,x), \dots \underline{cfg}_m(t,x)) \qquad \mathbf{c} = \mathbf{0}, \mathbf{1}$$

$$\mathbf{f}(x) = \underline{cfg}_1(\mathbf{T}(x), x)$$

[18] A constructor commands states, roughly, *store in a certain register the result of applying* $\mathbf{c}_i$ *to the current contents of certain registers*; a destructor command is analogous; and a branching command states *choose the next state among a given list of k states, according to the main constructor of the contents of a certain register.*

Then $(P, \mathbf{f})$ computes f (we stipulates that teh output of a RM is read off the first register).

To see that $(P, \mathbf{f})$ is provable in $\mathbf{RT}_o(\mathbb{W})$, let

$$\varphi[t] \equiv_{\mathrm{df}} W_0(\underline{cfg}_0(t,x)) \wedge \cdots \wedge W_0(\underline{cfg}_m(t,x)).$$

By Lemma 4.1 and $\bar{P}$ we have $\mathbf{RT}(\mathbb{W}) \vdash Cl_{\mathbb{W}}[\lambda t.\varphi]$. Thus, by induction on φ,

$$\mathbf{RT}_o(\mathbb{W}) \vdash W_1(t) \to \varphi[t].$$

Combining this with (6) we get

$$\mathbf{RT}_o(\mathbb{W}, P) \vdash W_\ell(x) \to W_0(\underline{cfg}_1(\mathbf{T}(x), x)),$$

that is, $(P, \mathbf{f})$ is provable. □

Remark: ℓ above can be taken to be $k+1$. In [Lei94c] a different applicative program $(P', \mathbf{f})$ for f is defined, for which only two levels suffice: $\mathbf{RT}_o(\mathbb{W}, P') \vdash W_1(w) \to W_0(\mathbf{f}(w))$.

From Theorem 4.2 we immediately conclude:

Corollary **4.3** *Every function Turing-computable in polynomial time is provable in* $\mathbf{RT}_o(\mathbb{W})$ *using quantifier-free induction.*

The same holds for all word algebras with 2 or more unary constructors. However, for word algebras with only one unary constructor, like $\mathbb{N}$, the relation between RM computation and Turing computation is slightly less direct, as we have the following (essentially due to Hartmanis and Gurevich; see [Lei94c] for a proof):

Lemma **4.4** *A numeric function f is computable on a RM over $\mathbb{N}$ in polynomial time iff it is computable on a multi-tape TM in linear space.*

Therefore, Theorem 4.2 implies in this case:

Corollary **4.5** *Every numeric function computable on a multi-tape TM in linear space is provable in* $\mathbf{RT}(\mathbb{N})$ *using quantifier-free induction.*

Using in the proof of 4.2 Lemma 3.8 in place of Lemma 3.6 we obtain:

Proposition **4.6** *Every function f over a word algebra $\mathbb{A}$ computable in Kalmar-elementary time is provable in* $\mathbf{RT}(\mathbb{A})$.

Note that induction is used here for universal formulas. From Theorem 5.2 below it follows that no additional functions are provable when induction is used for formulas of arbitrary quantifier complexity.[19]

[19] However, if only the first two ramification levels are permitted, then quantifier complexity of induction corresponds to iteration of exponentials, as we shall show elsewhere.

5 Complexity classes from ramified provability

5.1 Ramified proofs as ramified programs

Let $\mathbf{1\lambda}^{\times}$ be the typed lambda calculus with pairing. That is, given a set of base types, the types are generated by: if τ_1 and τ_2 are types then so are $\tau_1 \to \tau_2$ and $\tau_1 \times \tau_2$. Given a set of base terms (each assigned a type), terms are built from variables and base-terms using λ-abstraction, application, as well as by (i) *Pairing:* if M_1, M_2 are terms of types τ_1 and τ_2 respectively, then $\langle M_1, M_2\rangle$ is a term of type $\tau_1 \times \tau_2$; and (ii) *Projection:* if P is a term of type $\tau_1 \times \tau_2$ then $\mathbf{1}P$ and $\mathbf{2}P$ are terms of types τ_1 and τ_2, respectively.

Let $\mathbf{RRec}(\mathbb{A})$ be $\mathbf{1\lambda}^{\times}$, with base types $0, 1, \ldots$ (type ℓ is meant to denote $\mathbb{A}_\ell$), and with the following constants: (1) for each constructor $\mathbf{c}_i$ of $\mathbb{A}$ and each $\ell \geq 0$, a constant c_i^ℓ of type $\sigma_i[\ell]$; (2) for each type τ, a constant $\mathbf{R}_\tau^{\mathbb{A}} = \mathbf{R}_\tau$ of type $\ell, \sigma_1[\tau], \ldots \sigma_k[\tau] \to \tau$, where $\ell = 1+\max\{m \mid \ m \text{ occurs in } \tau\ \}$; (3) for each type τ, a constant $\mathbf{S}_\tau^{\mathbb{A}} = \mathbf{S}_\tau$ of type $o, \rho_1[\tau], \ldots \rho_k[\tau] \to \tau$, where $\rho_i[\tau] = o^{r_i} \to \tau$.

The reduction rules of $\mathbf{RRec}(\mathbb{A})$ are: (i) the β-rule; (ii) Pair-reductions: $\mathbf{1}\langle M_1, M_2\rangle \Rightarrow M_1$, $\mathbf{2}\langle M_1, M_2\rangle \Rightarrow M_2$; (iii) for each type τ the rule of *recurrence in type* τ; and (iv) for each type τ the rule of *selection in type* τ:

$$\mathbf{S}_\tau(c_i^0 a_1 \ldots a_{r_i}) M_1 \cdots M_k \ \Rightarrow\ M_i a_1 \ldots a_{r_i}$$

We write $\mathbf{RRec}_o(\mathbb{A})$ for the fragment of $\mathbf{RRec}(\mathbb{A})$ with $\mathbf{R}_\tau$ only for types τ without $\to$.

We now reformulate the mapping κ from $\mathbf{IT}(\mathbb{A}, P)$ to $\mathbf{Rec}(\mathbb{A})$ as a mapping from $\mathbf{RT}(\mathbb{A}, P)$ to $\mathbf{RRec}(\mathbb{A})$. The modifications are as follows:

2^ If Π derives φ by $\wedge$-introduction, then $\kappa\Pi = \langle \kappa\Pi_1,\ \kappa\Pi_2\rangle$.
3^ If Π derives φ by left-$\wedge$-elimination, then $\kappa\Pi = \mathbf{1}(\kappa\Pi_1)$; if Π derives φ by right-$\wedge$-elimination, then $\kappa\Pi = \mathbf{2}(\kappa\Pi_1)$.
5' If Π derives $A_\ell(\mathbf{c}_i(t_1 \ldots t_{r_i}))$ by the generative $\mathbf{c}_i$ rule, then $\kappa\Pi = c_i^\ell(\kappa\Pi_1) \cdots (\kappa\Pi_{r_i})$.
6' If Π derives $\varphi[\mathbf{t}]$ by $\mathbb{A}$-induction from $A_\ell(\mathbf{t})$ and $Cl_{\mathbf{c}_i}[\lambda u.\varphi]$ $(i = 1 \ldots k)$, then $\kappa\Pi = \mathbf{R}_{\kappa\varphi}(\kappa\Pi_1)(\kappa\Pi_2) \cdots (\kappa\Pi_{k+1})$.
7 If Π derives $\varphi[\mathbf{t}]$ by $\mathbb{A}$-selection from $A_\ell(\mathbf{t})$ and $Cl^{o}_{\mathbf{c}_i}[\lambda u.\varphi]$ $(i = 1 \ldots k)$, then $\kappa\Pi = \mathbf{S}_{\kappa\varphi}(\kappa\Pi_1)(\kappa\Pi_2) \cdots (\kappa\Pi_{k+1})$.

A ramified analog of Theorem 2.6 is proved now exactly as for the un-ramified case:

THEOREM **5.1** *Suppose that the equational program* $(P, \mathbf{f})$ *computes a function* f *over* $\mathbb{A}$. *If* Π *is a deduction of* $\mathbf{RT}(\mathbb{A}, P)$ *deriving* $A_j(x) \to A_\ell(\mathbf{f}(x))$, *then* $\kappa\Pi$ *represents* f *in* $\mathbf{RRec}(\mathbb{A})$. *Similarly, if* Π *is a deduction of* $\mathbf{RT}_o(\mathbb{A})$, *then* $\kappa\Pi$ *represents* f *in* $\mathbf{RRec}_o(\mathbb{A})$.

5.2 Characterization theorems

THEOREM **5.2** *A function over a free algebra $\mathbb{A}$ is provable in* $\mathbf{RT}(\mathbb{A})$ *iff it is computable in Kalmar-elementary time.*

Proof. A function over $\mathbb{A}$ provable in $\mathbf{RT}(\mathbb{A})$ is represented in $\mathbf{RRec}(\mathbb{A})$, by Theorem 5.1, and is therefore computable in elementary time by [Lei94a].

Conversely, if a function is computable in elementary time, then it is provable in $\mathbf{RT}(\mathbb{A})$ by Proposition 4.6. □

THEOREM **5.3** *A function over* $\mathbb{W}$ *is provable in* $\mathbf{RT}_o(\mathbb{W})$ *iff it is computable in polynomial time.*

Proof. If a function f over $\mathbb{W}$ is provable in $\mathbf{RT}_o(\mathbb{W})$ then it is represented in $\mathbf{RRec}_o(\mathbb{W})$, by Theorem 5.1, and is therefore computable in polynomial time by [Lei94c].

Conversely, if a function is computable in polynomial time, then it is provable in $\mathbf{RT}(\mathbb{W})$ by Proposition 4.3. □

Finally, we obtain in the same way

THEOREM **5.4** *A function over* N *is provable in* $\mathbf{RT}_o(\mathrm{N})$ *iff it is computable on a multi-tape Turing-machine in linear space.*

6 Discussion

We conclude with comments of potential interest to the philosophically inclined reader.

6.1 Ecto-algebraic theories

Peano's original axiomatization of arithmetic [Pea89] refers explicitly to a unary predicate N, intended to denote the property "is a natural number." His axioms fall into four groups: Generative axioms for N, which guarantee that the extension of N includes the denotations of all numerals; *Separation axioms:* $\forall x\ \neg(0 = \mathrm{s}(x))$ and $\forall x, y\ (\mathrm{s}(x) = \mathrm{s}(y) \rightarrow x = y)$, which guarantee that the denotations of the numerals are distinct; the defining equations for addition and multiplication; and the schema of Induction. The generative axioms are redundant in the traditional model theoretic approach to axiomatization, where the universe of discourse is assumed to be the structure in hand. This approach might be dubbed *endo-algebraic,* in contrast to Peano's approach, which is *ecto-algebraic* in the sense that the intended structure is delineated within broader universes.

Although ecto-algebraic theories have been viewed for almost a century as un-necessarily verbose, they remain of independent interest: they streamline reasoning about partial functions, support a natural homomorphism to typed functional programs, and permit a transparent representation of the predicative critique of Peano Arithmetic (see §6.4 below).

We have generalized Peano's approach to arbitrary free algebras, while dropping the separation axioms and the equational function definitions. Indeed, the separation axioms have no effect on convergence of computation (provided the computation model in hand does not use branching on equality, which functional programs indeed do not). And by refraining from function definitions we bring out the intrinsic nature of the algebra in hand, rather than its computational character when suitably extended; hence our choice of the phrase *intrinsic.* One could wonder, for instance, whether the set **Fnc(PA)** of provably recursive functions of **PA** is fundamentally related to the set N, since if only addition is used then the class of provably recursive functions is much reduced [Pre29]. Gödel's Theorem (2.8 above) puts doubts to rest, by linking **Fnc(PA)** solely to recurrence in finite type. Our Theorem 2.7 is the proof theoretic analog of that result.

6.2 Existence and divergence

A number of logics, dubbed *free logics,* have been proposed for incorporating partial functions and non-denoting terms into formal reasoning. Most of these use an *existence predicate* E, intended to hold true only of the denoting terms, with the rule of universal instantiation restricted accordingly: from $\forall x \varphi[x]$ *and* $E(\mathbf{t})$ infer $\varphi[\mathbf{t}]$. There is a choice as to the free variables: they can be interpreted as ranging over potentially non-existing entities, or over existing entities only.[20] Common to all these logics is the underlying semantics of a fixed universe, with the predicate E used to relate the scopes of free variables, of bound variables, and of terms.

Intrinsic theories, on the other hand, are based on standard logic, and have each a family of standard models, in which all terms are denoting. This is different from free logic in several respects. Convergence and denotation are treated here as mathematical issues, not logical ones, and they do not require any departure from traditional deductive systems. Computational divergence is reflected simply in the canonical model not being a model of the program.[21] It seems therefore plausible that intrinsic axiomatizations will prove to be more useful than free logics for reasoning about functional programs.

[20] The first approach is called *outer* or *Meinongian* in [Lam91], *E-logic* in [TvD88, I.2.2], and *logic of existence* in [Fef95]; it is developed in [Sco67, Sco79]. The latter is called *inner* or *Russelian* in [Lam91], *logic of partial terms* in [Bee85], *E^+-logic* in [TvD88, I.2.2], and *logic of definedness* in [Fef95].

[21] E-logic might be interpreted as an intrinsic theory, with all variables relativized to E, but that is of course a limitation on the expressiveness of the intrinsic theory.

6.3 Existential quantification

The definition of "provably recursive functions" uses the existential quantifier in an essential way, while the ecto-algebraic definition of "provable functions" does not. The difference is technically minor, but it continues a long history of doing away with $\exists$: from formalization styles (Hilbert's epsilon), to model theoretic results (Skolemization and Herbrand's Theorem), to the fact (observed in [CS95]) that work in applied mathematics and theoretical physics is mostly equational and quantifier-free.

6.4 Ramification

The ramification of intrinsic theories can be motivated on grounds analogous to the ramification of sets in predicative second order logic. The set N of natural numbers is implicitly defined by Peano's axioms: the generative axioms convey a lower bound on the extension of N, and the induction schema approximates an upper bound. However, as observed in [Nel86], if a formula φ has quantifiers, then its meaning presupposes the delineation of N as the domain of the quantifiers, and therefore using induction over φ as a component of the delineation of N is a circular enterprise. Our ecto-algebraic setting makes it possible to articulate this critique precisely and to address it directly: using induction over formulas that refer to the predicate N in order to delineate N itself is circular. To circumvent this circularity one stratifies N into a sequence N_0, $\mathrm{N}_1 \ldots$, just as in Predicative Analysis one avoids impredicative set quantification by a stratified progression of universes of sets.[22] Namely, N_0 is intended as a first approximation of N, with induction over N_0 permissible for formulas without explicit reference to N. Then, N_{i+1} is an approximation improving on N_i, since induction over N_{i+1} is allowed for additional formulas. Note, however, that the predicative critique does not apply to forms of induction that do not depend on N as a totality, notably the Selection schema.

A similar argument applies to any free algebra $\mathbb{A}$, leading to a ramificiation into $\mathbb{A}_0, \mathbb{A}_1 \ldots$.[23] The crucial bifurcation is between the first two levels, $\mathbb{A}_0$ and $\mathbb{A}_1$. Taking $\mathbb{A} = \mathbb{W}$ for instance, an element $w \in \mathbb{W}_0$ can be used as a bit-store: one can access the i'th bit of w, namely $\underline{case}\,(\underline{dstr}^{[i]}(w), \epsilon, 0, 1)$; but w cannot be used as a completed totality, i.e. as template for induction/recursion. The use of further levels $\mathbb{A}_2, \mathbb{A}_3, \ldots$ contributes neither to the proof theoretic power of **RT**($\mathbb{A}$), nor to the definitional power of **RRec**($\mathbb{A}$). However, it does contribute to succinctness (see [Lei94c] for ramified recurrence).[24]

[22] The initial universe U_0 consists of the first-order definable sets, $U_{\alpha+1}$ of the sets definable using set quantification over U_β with $\beta < \alpha$, and $U_\xi =_{\mathrm{df}} \bigcup_{\alpha<\xi} U_\alpha$ for limit ordinals ξ.

[23] The analogous ramification of recurrence is discussed and used in [Lei94c].

[24] Several authors have independently discovered the usefulness of data bifurcation [BC92, Lei90b, Sim88].

6.5 Feasibility

The ramification of intrinsic theories goes only some way in restricting their computational contents, as shown in Theorem 5.2. Indeed, while ramification excludes the circular definition of $\mathbb{A}$, $\mathbf{RT}(\mathbb{A})$ still reflects the admission of each level $\mathbb{A}_i$ as a completed totality: applying induction to a formula φ of $\mathbf{RT}(\mathbb{A})$ presupposes that the meaning of φ has been understood, but if $\mathbb{A}_i$ is referred to negatively in φ, as, for example, in $\forall v \; N_0(v) \rightarrow N_0(\mathrm{e}(v,u))$ (the formula used above in proving exponentiation), then the meaning of φ hinges on our understanding not only of what is *in* $\mathbb{A}_0$, but also of what is *not*.

It follows that a *"finitistic"* view of mathematical existence, which requires that infinite totalities be used only as unbounded processes but never as completed totalities, corresponds to $\mathbf{RT}_o(\mathbb{A})$. Thus, theorems 4.2 and 4.3 indicate a close kinship between feasible computing and a finitistic mathematical ontology. A similar kinship, between poly-time and an articulation of the finitistic position within second order logic, is discussed in [Lei94b]

References

[BC92] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions, 1992.

[Bee85] Michael Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, Berlin, 1985.

[CS95] Roland Chuaqui and Patrick Suppes. Free-variable axiomatic foundations of infinitesimal analysis: a fragment with finitary consistency proof. *Journal of Symbolic Logic*, 60:122–159, 1995.

[Fef95] Solomon Feferman. Definedness, 1995. Preprint.

[Fri78] H. Friedman. Classically and intuitionistically provable recursive functions. In G. H. Muller and D. S. Scott, editors, *Higher Set Theory*, pages 21–28. North-Holland, Amsterdam, 1978.

[Göd58] Kurt Gödel. Über eine bisher noch nicht benutzte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958.

[Hei67] J.van Heijenoort. *From Frege to Gödel, A Source Book in Mathematical Logic, 1879-1931.* Harvard University Press, Cambridge, MA, 1967.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, New York, 1980. Preliminary manuscript: 1969.

[Lam91] K. Lambert. Introduction. In K. Lambert, editor, *Philosophical applications of free logic*, Oxford and New York, 1991. Oxford University Press.

[Lei83] Daniel Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *Proceedings of the Twenty Fourth Annual Symposium on the Foundations of Computer Science*, pages 460–469, Washington, 1983. IEEE Computer Society.

[Lei85] Daniel Leivant. Syntactic translations and provably recursive functions. *Journal of Symbolic Logic*, 50:682–688, 1985.

[Lei90a] Daniel Leivant. Contracting proofs to programs. In P. Odifreddi, editor, *Logic and Computer Science*, pages 279–327. Academic Press, London, 1990.

[Lei90b] Daniel Leivant. Subrecursion and lambda representation over free algebras. In Samuel Buss and Philip Scott, editors, *Feasible Mathematics*, Perspectives in Computer Science, pages 281–291. Birkhauser-Boston, New York, 1990.

[Lei91] Daniel Leivant. Semantic characterization of number theories. In Y. Moschovakis, editor, *Logic from Computer Science*, pages 295–318. Springer-Verlag, New York, 1991.

[Lei94a] D. Leivant. Predicative recurrence in finite type. In A. Nerode and Yu.V. Matiyasevich, editors, *Logical Foundations of Computer Science (Third International Symposium)*, LNCS, pages 227–239, Berlin, 1994. Springer-Verlag.

[Lei94b] Daniel Leivant. A foundational delineation of poly-time. *Information and Computation*, 110:391–420, 1994. (Special issue of selected papers from LICS'91, edited by G. Kahn). Preminary report: A foundational delineation of computational feasibility, in Proceedings of the Sixth IEEE Conference on Logic in Computer Science, IEEE Computer Society Press, 1991.

[Lei94c] Daniel Leivant. Ramified recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffrey Remmel, editors, *Feasible Mathematics II*, Perspectives in Computer Science, pages 320–343. Birkhauser-Boston, New York, 1994.

[Nel86] Edward Nelson. *Predicative Arithmetic.* Princeton University Press, Princeton, 1986.

[Par77] Charles Parsons. On a number-theoretic choice schema and its relation to induction. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 459–473. North-Holland, Amsterdam, 1977.

[Pea89] Giuseppe Peano. *Arithmetices principia, novo methodo exposita.* Torino, 1889. English translation in [Hei67], 83–97.

[Pra71] Dag Prawitz. Ideas and results in proof theory. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 235–307, Amsterdam, 1971. North-Holland.

[Pre29] M. Presburger. Ueber die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen in welchem die addition als einzige operation hervortritt. In *Comptes Rendues, Ier Congrè des Mathématiques des Pays Salves*, pages 192–201,395, Warsaw, 1929.

[Sch24] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. English translation: On the building blocks of mathematical logic, in [Hei67], 355-366.

[Sco67] Dana Scott. Existence and description in formal logic. In *Bertrand Russell: Philosopher of the Century*, pages 28–48. Little, Brown and Co., Boston, 1967.

[Sco79] Dana Scott. Identity and existence in formal logic. In *Applications of sheaves*, LNM 753, pages 660–669. Springer-Verlag, Berlin, 1979.

[Sim88] Harold Simmons. The realm of primitive recursion. *Archive for Mathematical Logic*, 27:177–188, 1988.

[TvD88] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics, an Introduction.* North-Holland, Amsterdam, 1988. Two volumes.

On Herbrand's Theorem

Samuel R. Buss

University of California, San Diego
La Jolla, California 92093-0112, U.S.A.

Abstract. We firstly survey several forms of Herbrand's theorem. What is commonly called "Herbrand's theorem" in many textbooks is actually a very simple form of Herbrand's theorem which applies only to $\forall\exists$-formulas; but the original statement of Herbrand's theorem applied to arbitrary first-order formulas. We give a direct proof, based on cut-elimination, of what is essentially Herbrand's original theorem. The "no-counterexample theorems" recently used in bounded and Peano arithmetic are immediate corollaries of this form of Herbrand's theorem. Secondly, we discuss the results proved in Herbrand's 1930 dissertation.

1 Introduction

This paper discusses the famous theorem of Herbrand, which is one of the central theorems of proof-theory. The theorem called "Herbrand's theorem" in modern-day logic courses is typically only a very weak version of the theorem originally stated by Herbrand in his 1930 dissertation [8]. His 1930 dissertation contains in addition a number of other fundamental results, including, the unification algorithm, the fact that equality axioms do not help in proving equality-free sentences, a main result that is very similar to the cut-elimination theorem, and even a partial statement of the completeness theorem. The dissertation also contains a serious flaw in the proof of the main theorem, which was discovered and corrected by Dreben et al. in the 1960's, as well as earlier by Gödel in unpublished work.

This author first studied Herbrand's thesis while preparing an introductory article [1]; there we restate Herbrand's theorem in an essentially equivalent form and give a direct proof based on the cut-elimination theorem (this restatement is the same as Theorem 3 of the present paper). Since Herbrand's work contains a number of interesting constructions that are not widely known, we felt it worthwhile to prepare this paper as a survey of Herbrand's main results in chapter 5 of his dissertation.

The outline of this paper is follows: first we discuss the commonly used, weak form of Herbrand's theorem that applies only to $\forall\exists$-formulas. Then we discuss two ways of extending the theorem to general formulas: firstly, using Herbrand/Skolem functions to reexpress an arbitrary formula as a $\forall\exists$ formula, and, secondly, using a method based on "strong $\vee$-expansions" to prove a theorem

* Supported in part by NSF grant DMS-9205181

which is very similar to the fundamental theorem of Herbrand. We give proofs of these results based on the cut-elimination theorem for the sequent calculus. After that, we discuss the fundamental theorem as stated by Herbrand. We also discuss the relationship of Herbrand's work to the completeness theorem and the cut-elimination theorem. Finally, we briefly discuss the error in Herbrand's proof; for a full discussion of this error and its correction, the reader should refer to the papers by Dreben et al and to Goldfarb [7] for an account of Gödel's unpublished work.

References on Herbrand's dissertation include the dissertation itself [8], the translation of its fifth chapter and the accompanying notes by Dreben and van Heijenoort [9]. Discussions of the errors in Herbrand's thesis can be found in the papers by Dreben et al. [3,4,5] and in Goldfarb [7]. Herbrand's collected works are contained in [10,11]. Goldfarb [6] has further discussion of the history of Herbrand's theorem and an application to incompleteness.

All proofs in this paper are presented in terms of the sequent calculus; however for space reasons, background material and definitions for the sequent calculus are not included in this paper. A reader unfamiliar with the sequent calculus should either skip all proofs or refer to [1,17] for definitions.

We are grateful to R. Parikh and W. Goldfarb for comments on an earlier draft of this paper.

2 The weak form of Herbrand's theorem

Herbrand's theorem is one of the fundamental theorems of mathematical logic and allows a certain type of reduction of first-order logic to propositional logic. In its simplest form it states:

Theorem 1. *Let T be a theory axiomatized by purely universal formulas. Suppose that $T \vDash (\forall \mathbf{x})(\exists y_1, \ldots, y_k)B(\mathbf{x}, \mathbf{y})$ with $B(\mathbf{x}, \mathbf{y})$ a quantifier-free formula. There there is a finite sequence of terms $t_{i,j} = t_{i,j}(\mathbf{x})$, with $1 \le i \le r$ and $1 \le j \le k$ so that*

$$T \vdash (\forall \mathbf{x}) \left(\bigvee_{i=1}^{r} B(\mathbf{x}, t_{i,1}, \ldots, t_{i,k}) \right).$$

It is well-known how to give a model-theoretic proof of Theorem 1; it is also straightforward to give a constructive, proof-theoretic proof based on the cut-elimination theorem as follows:

Proof. Since T is axiomatized by purely universal formulas, it may, without loss of generality, be axiomatized by quantifier-free formulas (obtained by removing the universal quantifiers). Let $\mathfrak{T}$ denote the set of sequents of the form $\longrightarrow A$ with A a (quantifier-free) axiom of T. Define a $LK_{\mathfrak{T}}$ proof to be a sequent calculus proof in Gentzen's system LK, except allowing sequents from $\mathfrak{T}$ in addition to the usual initial sequents.[3] Since $T \vDash (\forall \mathbf{x})(\exists \mathbf{y})B(\mathbf{x}, \mathbf{y})$, there is a $LK_{\mathfrak{T}}$-proof of the sequent $\longrightarrow(\exists \mathbf{y})B(\mathbf{a}, \mathbf{y})$.

[3] $LK_{\mathfrak{T}}$ may optionally contain equality axioms as initial sequents.

By the free-cut elimination theorem, there is a free-cut free $LK_{\mathfrak{T}}$-proof P of this sequent, and since the $\mathfrak{T}$-sequents contain only quantifier-free formulas, all cut formulas in P are quantifier-free. Thus, any non-quantifier-free formula in P must be of the form $(\exists y_j)\cdots(\exists y_k)B(\mathbf{a},t_1,\ldots,t_{j-1},y_j,\ldots,y_k)$ with $1 \leq j < k$. We claim that P can be modified to be a valid proof of a sequent of the form

$$\rightarrow B(\mathbf{a},t_{1,1},\ldots,t_{1,k}),\ldots,B(\mathbf{a},t_{r,1},\ldots,t_{r,k}).$$

The general idea is to remove all $\exists$*:right* inferences in P and remove all existential quantifiers, replacing the bound variables by appropriate terms. Since there may have been contractions on existential formulas that are no longer identical after terms are substituted for variables it will also be necessary to remove contractions and add additional formulas to the sequents. To do this more formally, we know that any sequent in P is of the form $\Gamma{\rightarrow}\Delta,\Delta'$ (up to order of the formulas in the sequent), where each formula in Γ and Δ is quantifier-free and where each formula in Δ' is not quantifier-free but is purely existential. We can then prove by induction on the number of lines in the free-cut free proof of $\Gamma{\rightarrow}\Delta,\Delta'$ that there is an $r \geq 0$ and a cedent Δ'' of the form

$$B(\mathbf{a},t_{1,1},\ldots,t_{1,k}),\ldots,B(\mathbf{a},t_{r,1},\ldots,t_{r,k})$$

such that $\Gamma{\rightarrow}\Delta,\Delta''$ is provable. We leave the rest of the details to the reader. □

We define an *instance* of a universal formula $(\forall\mathbf{x})A(\mathbf{x})$ to be any quantifier-free formula $A(\mathbf{t})$. It is not hard to see using cut elimination, that if a quantifier-free formula C is a consequence of a universal theory T, then it is a tautological consequence of some finite set of instances of axioms of T and of equality axioms. In the special case where T is the null theory, we have that C is a consequence of instances of equality axioms (and C is therefore called a *quasitautology*). If, in addition, C does not involve equality, C will be tautologically valid. Thus, Herbrand's theorem reduces provability in first-order logic to generation of (quasi)tautologies.

The weak form of Herbrand's theorem stated above as Theorem 1 has limited applicability since it applies only to $\forall\exists$-consequences of universal theories: fortunately, however, there are several ways to extend Herbrand's theorem to more general situations. In section 3 below, we explain one such generalization; but first we give a simpler method of widening the applicability of Herbrand's theorem, based on the introduction of new function symbols, which we call *Herbrand* and *Skolem* functions, that allow quantifier alternations to be reduced.

For notational simplicity, we will consider only formulas in prenex normal form for the rest of this section; however, the definitions and theorem below can be readily generalized to arbitrary formulas.

Definition 2. Let $(\exists x)A(x,\mathbf{c})$ be a formula with $\mathbf{c}$ all of its free variables. The *Skolem function* for $(\exists x)A$ is represented by a function symbol $f_{\exists xA}$ and has the defining axiom:

$$\textit{Sk-def}(f_{\exists xA}): \quad (\forall\mathbf{y})(\forall x)\,(A(x,\mathbf{y}) \rightarrow A(f_{\exists xA}(\mathbf{y}),\mathbf{y}))\,.$$

Note that *Sk-def*($f_{\exists xA}$) implies $(\forall \mathbf{y})\,((\exists x)A(x,\mathbf{y}) \leftrightarrow A(f_{\exists xA}(\mathbf{y}),\mathbf{y}))$.

Definition 3. Let $A(\mathbf{c})$ be a formula in prenex form. The *Skolemization*, $A^S(\mathbf{c})$, of A is the formula defined inductively by:

(1) If $A(\mathbf{c})$ is quantifier-free, then $A^S(\mathbf{c})$ is $A(\mathbf{c})$.
(2) If $A(\mathbf{c})$ is $(\forall y)B(\mathbf{c},y)$, then $A^S(\mathbf{c})$ is the formula $(\forall y)B^S(\mathbf{c},y)$.
(3) If $A(\mathbf{c})$ is $(\exists y)B(\mathbf{c},y)$, then $A^S(\mathbf{c})$ is $B^S(\mathbf{c},f_A(\mathbf{c}))$, where f_A is the Skolem function for A.

It is a simple, but important fact that $A^S \vDash A$.

The *Skolemization* of a theory T is the theory $T^S = \{A^S : A \in T\}$. Note that T^S is a purely universal theory. Incidentally, the set of *Sk-def* axioms of the Skolem functions can be equivalently expressed as a set of universal formulas; however, they are not included in theory T^S. From model-theoretic considerations, it is not difficult to see that T^S contains and is conservative over T.

We next define the concept of 'Herbrandization' which is completely dual to the notion of Skolemization:

Definition 4. Let $(\forall x)A(x,\mathbf{c})$ be a formula with $\mathbf{c}$ all of its free variables. The *Herbrand function* for $(\forall x)A$ is represented by a function symbol $h_{\forall xA}$ and has the defining axiom:

$$(\forall \mathbf{y})(\forall x)\,(\neg A(x,\mathbf{y}) \rightarrow \neg A(h_{\forall xA}(\mathbf{y}),\mathbf{y}))\,.$$

Note that this implies $(\forall \mathbf{y})\,((\forall x)A(x,\mathbf{y}) \leftrightarrow A(h_{\forall xA}(\mathbf{y}),\mathbf{y}))$. The Herbrand function can also be thought of as a 'counterexample function'; in that $(\forall x)A(x)$ is false if and only if $h_{\forall xA}$ provides a value x which is a counterexample to the truth of $(\forall x)A$.

Definition 5. Let $A(\mathbf{c})$ be a formula in prenex form. The *Herbrandization*, $A^H(\mathbf{c})$, of A is the formula defined inductively by:

(1) If $A(\mathbf{c})$ is quantifier-free, then $A^H(\mathbf{c})$ is $A(\mathbf{c})$.
(2) If $A(\mathbf{c})$ is $(\exists y)B(\mathbf{c},y)$, then $A^H(\mathbf{c})$ is the formula $(\exists y)B^H(\mathbf{c},y)$.
(3) If $A(\mathbf{c})$ is $(\forall y)B(\mathbf{c},y)$, then $A^H(\mathbf{c})$ is $B^H(\mathbf{c},h_A(\mathbf{c}))$, where h_A is the Herbrand function for A.

It is not hard to see that $A \vDash A^H$. Note that A^H is purely existential.

Theorem 6. *Let T be set of prenex formulas and A any prenex formula. Then the following are equivalent:*

(1) $T \vDash A$,
(2) $T^S \vDash A$,
(3) $T \vDash A^H$,
(4) $T^S \vDash A^H$,

This theorem is easily proved from the above definitions and remarks. The importance of Theorem 6 lies in the fact that T^S is a universal theory and that A^H is an existential formula, and that therefore Herbrand's theorem applies to $T^S \models A^H$. Thus, Theorem 6 allows Theorem 1 to be applied to an arbitrary logical implication $T \models A$, at the cost of converting formulas to prenex form and introducing Herbrand and Skolem functions.

3 A strong form of Herbrand's theorem

Herbrand actually proved a much more general theorem than Theorem 1 which applies directly whenever $\models A$, for A a general formula, not necessarily $\forall\exists$. His result also avoids the use of Skolem/Herbrand functions. The theorem we state next is quite similar in spirit and power to the theorem as stated originally by [8].

In this section, we shall consider a first-order formula A such that $\models A$. Without loss of generality, we shall suppose that the propositional connectives in A are restricted to be $\land$, $\lor$ and $\neg$, and that the $\neg$ connective appears only in front of atomic subformulas of A. (The only reason for this convention is that it avoids having to keep track of whether quantifiers appear positively and negatively in A.)

Definition 7. Let A satisfy the above convention on negations. An $\lor$-*expansion* of A is any formula that can be obtained from A by a finite number of applications of the following operation:

(α) If B is a subformula of an $\lor$-expansion A' of A, replacing B in A' with $B \lor B$ produces another $\lor$-expansion of A.

A *strong* $\lor$-*expansion* of A is defined similarly, except that now the formula B is restricted to be a subformula with outermost connective an existential quantifier.

Definition 8. Let A be a formula. A *prenexification* of A is a formula obtained from A by first renaming bound variables in A so that no variable is quantified more than once in A and then using prenex operations to put the formula in prenex normal form.

Note that there will generally be more than one prenexification of A since prenex operations may be applied in different orders resulting in a different order of the quantifiers in the prenex normal form formula.

Definition 9. Let A be a valid first-order formula in prenex normal form, with no variable quantified twice in A. If A has $r \geq 0$ existential quantifiers, then A is of the following form with B quantifier-free:

$$(\forall x_1 \cdots x_{n_1})(\exists y_1)(\forall x_{n_1+1} \cdots x_{n_2})(\exists y_2) \cdots (\exists y_r)(\forall x_{n_r+1} \cdots x_{n_{r+1}})B(\mathbf{x}, \mathbf{y})$$

with $0 \leq n_1 \leq n_2 \leq \cdots \leq n_{r+1}$. A *witnessing substitution* for A is a sequence of terms (actually, semiterms) $t_1, \ldots t_r$ such that (1) each t_i contains arbitrary free variables but only bound variables from $x_1, \ldots, x_{n_i}$ and (2) the formula

$B(\mathbf{x}, t_1, \ldots, t_r)$ is a quasitautology (i.e., a tautological consequence of instances of equality axioms only). In the case where B does not contain the equality sign, then (2) is equivalent to B being a tautology.

Let T be a first-order theory. A sequence of terms is said to *witness A over T* if the above conditions hold except with condition (2) replaced by the weaker condition that $T \vDash (\forall \mathbf{x}) B(x, \mathbf{t})$.

Definition 10. A *Herbrand proof* of a first-order formula A consists of a prenexification A^* of a strong $\vee$-expansion of A plus a witnessing substitution σ for A^*.

A *Herbrand T-proof* of A consists of a prenexification A^* of a strong $\vee$-expansion of A plus a substitution which witnesses A over T.

We are now in a position to state the general form of Herbrand's theorem:

Theorem 11. *A first-order formula A is valid if and only if A has a Herbrand proof. More generally, if T is a universal theory, then $T \vDash A$ if and only if A has a Herbrand T-proof.*

Proof. We shall sketch a proof of only the first part of the theorem since the proof of the second part is almost identical. Of course it is immediate from the definitions that if A has a Herbrand proof, then A is valid. So suppose A is valid, and therefore has a cut-free LK-proof P. We shall modify P in stages so as to extract a Herbrand proof of P.

The first stage will involve restricting the formulas which can be combined by a contraction inference. In order to properly keep track of contractions of formulas in a sequent calculus proof, we must be careful to formulate inference rules with two hypotheses in a "multiplicative" fashion so as to avoid the problem of having implicit contractions on side formulas in inferences with two hypothesis such as $\vee$*:left* and $\wedge$*:right*. For example, we want to formulate the $\vee$*:left* inference rule in the multiplicative form

$$\frac{A, \Gamma \rightarrow \Delta \qquad B, \Gamma' \rightarrow \Delta'}{A \vee B, \Gamma, \Gamma' \rightarrow \Delta, \Delta'}$$

rather than in the "additive" form

$$\frac{A, \Gamma \rightarrow \Delta \qquad B, \Gamma \rightarrow \Delta}{A \vee B, \Gamma \rightarrow \Delta}$$

since the additive form contains implicit contractions on side formulas in Γ and Δ, whereas the multiplicative formulation does not contain implicit contractions. We also use analogous multiplicative formulations of the $\wedge$*:right* and cut rule. Of course, using multiplicative formulations rules instead of additive formulation does not change the strength of the sequent calculus, since either form may be derived from the other with the use of weak structural inferences. Furthermore, the cut-elimination and free-cut elimination theorems hold with either formulation. We therefore henceforth use the multiplicative formulation of the rules of inference for the sequent calculus.

A contraction inference is said to be a *propositional contraction* (resp., an $\exists$-*contraction* provided that the principal formula of the contraction is quantifier-free (resp., its outermost connective is an existential quantifier). The first step in modifying P is to form a cut-free proof P_1, also with endsequent $\rightarrow A$ such that all contraction inferences in P_1 are propositional or $\exists$-contractions. The construction of P_1 from P is done by a "contraction-elimination" procedure. For this purpose, we define the *E-depth* of a formula by letting the E-depth of a quantifier-free formula or a formula which begins with an existential quantifier be equal to zero, and defining the E-depth of other formulas inductively by letting the E-depth of $\neg\varphi$ equal the E-depth of φ plus one and by letting the E-depths of $\varphi \lor \psi$ and $\varphi \land \psi$ equal one plus the maximum of the E-depths of φ and ψ. Then we prove, by double induction on the maximum E-depth d of contraction formulas and the number of contractions of formulas of this maximum E-depth, that P_1 can be transformed into a proof in which all contractions are on formulas of E-depth zero. The induction step consists of removing a topmost contraction inference of the maximum E-depth d. For example, suppose that the following inference is a topmost contraction with principal formula of E-depth d;

$$\begin{array}{c} \ddots\vdots\cdot^{\cdot} R \\ \Gamma \rightarrow \Delta, (\forall x)B, (\forall x)B \\ \hline \Gamma \rightarrow \Delta, (\forall x)B \end{array}$$

Since P_1 is w.l.o.g. in free variable normal form and since this is a topmost contraction of E-depth d, we can modify the subproof R of P_1 by removing at most two $\forall$*:right* inferences and/or changing some *Weakening:right* inferences to get a proof of $\Gamma \rightarrow \Delta, B(a), B(a')$, where a and a' are free variables not appearing in the endsequent of R. Further replacing a' everywhere by a gives a proof of $\Gamma \rightarrow \Delta, B(a), B(a)$: we use this get to a proof ending:

$$\begin{array}{c} \ddots\vdots\cdot^{\cdot} \\ \Gamma \rightarrow \Delta, B(a), B(a) \\ \hline \Gamma \rightarrow \Delta, B(a) \\ \hline \Gamma \rightarrow \Delta, (\forall x)B \end{array}$$

Thus we have reduced the E-depth of the contraction inference. A similar procedure works for contractions of E-depth d with outermost connective a propositional connective—we leave the details to the reader. Note that the construction of P_1 depends on the fact that propositional inferences and $\forall$*:right* inferences can be pushed downward in the proof. It is not generally possible to push $\exists$*:right* inferences downward in a proof without violating eigenvariable conditions.

The second step in modifying P is to convert P_1 into a cut-free proof P_2 of some strong $\lor$-expansion A' of A such that every contraction in P_2 is propositional. This is done by the simple expedient of replacing every $\exists$-contraction in P_1 with an $\lor$*:right* inference, and then making the induced changes to all descendents of the principal formula of the inference. More precisely, starting with a lowermost $\exists$-contraction in P_1, say

$$\frac{\Gamma \rightarrow \Delta, (\exists x)B, (\exists x)B}{\Gamma \rightarrow \Delta, (\exists x)B}$$

replace this with an $\vee$*:left* inference

$$\frac{\Gamma \rightarrow \Delta, (\exists x)B, (\exists x)B}{\Gamma \rightarrow \Delta, (\exists x)B \vee (\exists x)B}$$

and then, in order to get a syntactically correct proof, replace, as necessary, subformulas $(\exists x)B'$ of formulas in P with $(\exists x)B' \vee (\exists x)B'$ (we use the notation B' since terms in B may be different in its descendents). Iterating this process yields the desired proof P_2 of a strong $\vee$-expansion A' of A. By renaming bound variables in P_2 we can assume w.l.o.g. that no variable is quantified twice in any single sequent in P_2.

Thirdly, from P_2 we can construct a prenexification A^* of A' together with a witnessing substitution, thereby obtaining a Herbrand proof of A. To do this, we iterate the following procedure for pulling quantifiers to the front of the proved formula. Find any lowest quantifier inference in P_2 which has not already been handled: this quantifier inference corresponds to a unique quantifier, (Qx), appearing in the endsequent of the proof (and conversely, each quantifier in the endsequent of the proof corresponds to a unique quantifier inference, since all contraction formulas are quantifier-free). Use prenex operations to pull (Qx) as far to the front of the endsequent formula as possible (but not past the quantifiers that have already been moved to the front of the endsequent formula). Also, push the quantifier inference downward in the proof until it reaches the group of quantifier inferences that have already been pushed downward in the proof. It is straightforward to check that this procedure preserves the property of having a syntactically valid proof. When we are done iterating this procedure, we obtain a proof P_3 of a prenexification $\rightarrow A^*$ of A. It remains to define a witnessing substitution for A^*: this is now easy, for each existential quantifier $(\exists y_i)$ in A^*, find the corresponding $\exists$*:right* inference

$$\frac{\Gamma \rightarrow \Delta, B(t_i)}{\Gamma \rightarrow \Delta, (\exists y_i)B(y_i)}$$

and let the term t_i be from this inference. That this is a witnessing substitution for A^* is easily proved by noting that by removing the $\exists$*:right* inference from P_3, a proof of $A^*_M(\mathbf{x}, \mathbf{t})$ is obtained where A^*_M is the quantifier-free portion of A^*.

□

The above theorem can be used to obtain the following 'no-counterexample interpretation' which has been very useful recently in the study of bounded arithmetic (see [12,2,18]).[4]

Corollary 12. *Let T be a universal theory and suppose $T \vDash (\exists x)(\forall y)A(x, y, \mathbf{c})$ with A a quantifier-free formula. There is a $k > 0$ and terms $t_1(\mathbf{c})$, $t_2(\mathbf{c}, y_1)$,*

[4] This corollary is named after the more sophisticated no-counterexample interpretations of [13,14].

$t_3(\mathbf{c}, y_1, y_2), \ldots, t_k(\mathbf{c}, y_1, \ldots y_{k-1})$ *such that*

$$\begin{aligned} T \vDash & (\forall y_1)[A(t_1(\mathbf{c}), y_1, \mathbf{c}) \\ & \lor (\forall y_2)[A(t_2(\mathbf{c}, y_1), y_2, \mathbf{c}) \\ & \lor (\forall y_3)[A(t_3(\mathbf{c}, y_1, y_2), y_3, \mathbf{c}) \\ & \lor \cdots \lor (\forall y_k)[A(t_k(\mathbf{c}, y_1, \ldots, y_{k-1}), y_k, \mathbf{c}))] \cdots]]] \end{aligned}$$

To prove the corollary, note that the only strong $\lor$-expansions of A are formulas of the form $\bigvee(\exists x)(\forall y)A(x, y, \mathbf{c})$ and apply the previous theorem.

4 No recursive bounds on number of terms

It is interesting to ask whether it is possible to bound the value of r in Theorem 1. For this, consider the special case where the theory T is empty, so that we have an LK-proof P of $(\exists x_1, \ldots, x_k)B(\mathbf{a}, \mathbf{x})$ where B is quantifier-free. There are two ways in which one might wish to bound the number r needed for Herbrand's theorem: as a function of the size of P, or alternatively, as a function of the size of the formula $(\exists \mathbf{x})B$. For the first approach, it follows immediately from the proof of the cut-elimination theorem in [1] and the proof of Herbrand's theorem, that $r \leq 2^{||P||}_{2||P||}$, where 2^x_i is defined inductively by $2^x_0 = x$ and $2^x_{i+1} = 2^{2^x_i}$ and where $||P||$ equals the number of strong inferences in P. For the second approach, we shall sketch a proof below that r can not be recursively bounded as a function of the formula $(\exists \mathbf{x})B$. The proof is based on the unification algorithm contained in Herbrand [9, para. 2.4]

To show that r cannot be recursively bounded as a function of $(\exists \mathbf{x})B$, we shall prove that having a recursive bound on r would give a decision procedure for determining if a given existential formula is valid. Since it is well known that validity of existential first-order formulas is undecidable; this implies that r cannot be recursively bounded in terms of the formula size.

What we shall show is that, given a formula B as in Theorem 1 and given an $r > 0$, it is decidable whether there are terms $t_{1,1}, \ldots, t_{r,k}$ which make the formula

$$\bigvee_{i=1}^{r} B(\mathbf{a}, t_{i,1}, \ldots, t_{i,k}) \tag{1}$$

a tautology. (This fact was first proved by Herbrand by the same argument that we sketch here.) This will suffice to show that r cannot be recursively bounded. The quantifier-free formula B is expressible as a Boolean combination $C(D_1, \ldots, D_\ell)$ where each D_j is an atomic formula and $C(\cdots)$ is a propositional formula. If the formula (1) is a tautology, it is by virtue of certain formulas $D_j(\mathbf{a}, t_{i,1}, \ldots, t_{i,k})$ being identical. That is to say there is a finite set X of equalities of the form

$$D_j(\mathbf{a}, t_{i,1}, \ldots, t_{i,k}) \;=\; D_{j'}(\mathbf{a}, t_{i',1}, \ldots, t_{i',k})$$

such that, any set of terms $t_{1,1}, \ldots, t_{r,k}$ which makes all the equalities in X true will make (1) a tautology.

But now the question of whether there exist terms $t_{1,1}, \ldots, t_{r,k}$ which satisfy such a finite set X of equations is easily seen to be a first-order unification problem. The algorithm for solving first-order unification problems is given in Herbrand's thesis and is now-a-days well-known; Robinson [16] gives a method of getting a most general solution, and Paterson-Wegman [15] give a linear-time algorithm for unification. This algorithm either determines that no choice of terms will satisfy all the equations in X or will find a (most general) set of terms that satisfy the equations of X.

Since, for a fixed $r > 0$, there are only finitely many possible sets X of equalities, we have the following algorithm for determining if there are terms which make (1) a tautology: for each possible set X of equalities, check if it has a solution (i.e., a most general unifier), and if so, check if the equalities are sufficient to make (1) a tautology. □

5 The actual theorem of Herbrand

In this final section, we discuss the results contained in chapter 5 of Herbrand's Ph.D. thesis. The fundamental theorem of this chapter is very similar to Theorem 3 but differs in some details. We also describe the two proof systems, now called Q_H and Q'_H, that Herbrand used. The results stated by Herbrand include a version of the cut-elimination theorem and his proof methods give (or nearly give) a version of the completeness theorem. There was also a fairly serious error in Herbrand's proof, which was first described in published material by Dreben et al.; this error was apparently also recognized by Bernays in the 1930's and was discovered and corrected by Gödel in unpublished notes (see [7]). These errors in no way detract from the importance of Herbrand's work, since alternative proofs could be given. In any event, although there are some false lemmas in Herbrand's work, his main theorems are all fully correct.

5.1 Herbrand's fundamental theorem.

Herbrand's fundamental theorem applied to arbitrary first-order formulas A; in particular, A need not be in prenex normal form. By renaming variables, one can assume that no variable is quantified more than once in A and that no variable occurs both free and bound in A. Herbrand took prenex operations as fundamental in his proof theory (see the definitions of Q_H and Q'_H below). His formal system allowed prenex operations to be applied not only in a 'forward' direction which brings quantifiers to the front of a formula, but also in a 'reverse' direction pushing quantifiers further inward in a formula. Herbrand noted that for every formula A there is a unique formula, called the *canonical form* of A, which is obtained by applying prenex operations to subformulas of A to push quantifiers as far inward as possible. Let M be obtained from A by erasing all quantifiers from A. Since we use only connectives $\neg$, $\vee$ and $\wedge$ (Herbrand used only the first two) and because of our conventions on not reusing variables, it is easy to see that any prenex formula obtained from A by prenex operations consists of M preceded by a string of quantifiers. Thus all prenexifications of A differ only in the order of the quantifiers.

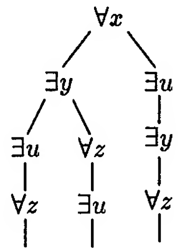
We now describe a *tree expansion of* A to consist of a finite set (also called a forest) of labeled trees: each tree has its leaves labeled with the formula M and has its internal nodes labeled with quantifiers $(\exists x)$ or $(\forall x)$ which already appear in A. Furthermore, the following properties should hold:

1. For any simple path from a root of a tree to a leaf, if the labels on the path are concatenated, then one obtains a formula which is equivalent to A and is obtainable from A by prenex operations only.
2. The trees are finite in that each internal node has only finitely many children.
3. If a node has more than one child, then none of its children are labeled with universal quantifiers.[5]

To given an example, consider a formula of the form

$$(\forall x)[(\exists y)(\forall z)A(x,y,z) \vee (\exists u)B(x,u)].$$

One possible set of trees associated to this formula is:



where M is $A(x,y,z) \vee B(x,u)$. Herbrand used a tabular notation to represent this situation; namely, for this example, he would write

$$\begin{array}{l} +x\ -u\ -y\ +z \\ +x\ -y\ -u\ +z \\ +x\ -y\ +z\ -u \end{array}$$

using $+x$ to mean $(\forall x)$ and $-u$ to mean $(\exists y)$, etc. Note that each line in the table corresponds to a path in the tree. To make the tree structure clearer, Herbrand then rewrites the table above as:

$$+x\begin{cases} -u \quad -y \quad +z \\ -y\begin{cases} -u \quad +z \\ +z \quad -u \end{cases} \end{cases}$$

[5] This optional condition is given in Herbrand's paragraph 2.33 of chapter 5 of his thesis. It is the analogue of our use of *strong* $\vee$-expansions in place of $\vee$-expansions.

The concept of a *proposition derived from A* is defined as follows: for each node in the tree assign a formula as follows: assign the matrix M to every leaf node, and assign to an internal node α labels with (Qv) the formula

$$(Qv)[P_1 \vee ... \vee P_n]$$

where $P_1, \ldots, P_m$ are the formulas assigned to the n children of α. Finally, take the disjunction of the formulas assigned to all the roots of trees in the forest, then rename variables so that no variable is used twice in this disjunction and form an arbitrary prenexification of this disjunction; the result is called a *proposition derived from A*.

It is clear that a proposition derived from A is equivalent to A, since it is obtained by using only the following types of operations: (a) prenex operations, (b) variable renamings, and (c) replacing subformulas Z with $Z \vee Z$ (i.e., $\vee$-expansion steps). Herbrand's fundamental theorem can now be stated as follows (the theory Q_H is described below; since it is sound and complete, A is Q_H-provable iff A is valid):

Theorem 13. *A is provable in the theory Q_H iff there is a proposition derived from A which has a witnessing substitution.*

5.2. Herbrand's proof systems

Herbrand's thesis primarily used a proof system which we shall denote Q_H; he also used a modified version, Q'_H, and his fundamental theorem states that provability in Q_H is equivalent to provability in Q'_H. Formulas in these proof systems involve the logical connectives $\neg$, $\vee$, $\forall$ and $\exists$; other symbols, such as $\rightarrow$ are abbreviations for more complex formulas. It is not permitted for a variable to be quantified twice in a formula, or to appear both free and bound in a formula. The system Q_H has all tautologies as axioms and has the following rules of inference:

1. Modus Ponens; from A and $A \rightarrow B$, infer B.
2. Rule of Simplification: If Z' is an alphabetic variant of Z, then Z may be inferred from $Z \vee Z'$.
3. Universal Generalization: from Φ, infer $(\forall x)\Phi$.
4. Existential Instantiation: from $\Phi(t)$, infer $(\exists x)\Phi(x)$.
5. The Rules of Passage: consider the following six pairs of logically equivalent formulas:

$$\begin{aligned}
\neg\forall x\Phi &\Leftrightarrow \exists x(\neg\Phi)\\
\neg\exists x\Phi &\Leftrightarrow \forall x(\neg\Phi)\\
(\forall x\Phi) \vee Z &\Leftrightarrow \forall x(\Phi \vee Z)\\
Z \vee (\forall x\Phi) &\Leftrightarrow \forall x(Z \vee \Phi)\\
(\exists x\Phi) \vee Z &\Leftrightarrow \exists x(\Phi \vee Z)\\
Z \vee (\exists x\Phi) &\Leftrightarrow \exists x(Z \vee \Phi)
\end{aligned}$$

There are twelve *rules of passage*; these allow a formula B to be inferred from the formula A provided B is obtained from A by replacing an occurrence of a subformula in A which is in one of the above twelve forms with the equivalent subformula given in the above table. (Note that the conventions on variable usage imply that x does not appear in Z.)

Herbrand's second proof system, which we call Q'_H, is obtained from Q_H by disallowing the rule of modus ponens, and replacing the rule of simplification by the *generalized rule of simplification* which permits B to be inferred from A when B is obtained from A by replacing a subformula of the form $Z \vee Z'$ with the subformula Z, provided Z' is an alphabetic variant of Z.

A corollary of Herbrand's fundamental theorem is the statement that a formula is Q_H-provable if and only if it is Q'_H-provable. This is a very intriguing fact, since it is evident that Q'_H is very similar to a cut-free sequent calculus proof system; in particular, there is an analogue of the subformula property of the sequent calculus which holds for Q'_H; namely, if one measures the complexity of formula in terms of the depth of quantifier nesting in the canonical form of a formula, then it is evident that all the formulas which appear in a Q'_H-proof of a formula A have complexity no greater than the complexity of A. Gentzen's paper on LK and cut-elimination appeared only four years later in 1934. However, we are reluctant to ascribe much of the credit for the cut-elimination theorem to Herbrand for two reasons: firstly, Q'_H does not have the elegance of the sequent calculus LK, and secondly, the errors in Herbrand's proof impinge directly on the proof of the equivalence of Q_H and Q'_H.

Indeed, it is precisely at the step of "elimination of modus ponens", which is the analogue of cut-elimination, that the errors in Herbrand's proof occur (see paragraph 5.3, lemma 3, chapter 5 of Herbrand's thesis). It is well-known that the process of cut-elimination in first-order logic leads to superexponential growth rates; however, in his erroneous proof, Herbrand claimed that much lower growth rates sufficed. The corrected versions of Herbrand's proof, given by Gödel (see [7]) and by Dreben et al. [3,4,5] do give superexponential growth rates that are similar to the growth rates known to hold for the cut-elimination theorem; and these growth rates are (nearly) optimal.

5.3. The completeness theorem.

Herbrand's thesis also includes a construction that is very close to the completeness theorem. (Recall that the completeness theorem was first proved by Gödel in 1930, in the same year that Herbrand's thesis was completed.) In his thesis, Herbrand discusses that fact that if there is no witnessing substitution for a proposition derived from A (as in Theorem 13), then it is possible to construct an sequence of finite domains where appropriate translations of A are false. Herbrand also discusses the possibility of having an infinite domain where A would be false in the usual sense; had he actually done this, he would have proved the completeness theorem. Somewhat surprisingly, Herbrand evidently knew that such an infinite domain could be obtained, but because of his constructive outlook, he declined to carry out the proof that such an infinite domain existed. Indeed he says

> "but only a 'principle of choice' could lead us to take a fixed system of values in an infinite domain."[6]

By this he means that it would be necessary to use the axiom of choice to obtain an infinite model in which A is false under the usual Tarskian semantics.

It is interesting to speculate why Herbrand chose not to state the completeness theorem. Firstly, Herbrand took a very strong constructive, formalist point of view, and he would have rejected non-constructive arguments on philosophical grounds. Indeed, Herbrand defined "true" to mean "provable in Q_H" rather than "true in all possible structures". Secondly, it seems that Herbrand felt that his fundamental theorem was of greater interest than a model-theoretic completeness theorem.

The issue of the completeness theorem has also some bearing on the status of the errors in Herbrand's thesis. The errors in his proof affected only the proof-theoretic results, and the completeness theorem, which Herbrand *could* have stated and proved, would not have been affected by these errors. Therefore, Herbrand could have obtained a alternative and correct proof of his fundamental theorem by using the following argument: suppose A is a formula and there is no proposition derived from A which has a witnessing substitution; then by the completeness theorem, there is an infinite domain (i.e., structure) where A is false; therefore, since the proof system Q_H is sound, there is no Q_H-proof of A. This argument proves the contrapositive of Theorem 13 and is thereby an error-free proof of Herbrand's fundamental theorem. Of course, this proof uses non-constructive methods and presumably would not have been attractive to Herbrand.

References

1. S. R. Buss, *An introduction to proof theory.* Typeset manuscript, to appear in *Handbook of Proof Theory*, 199?
2. ———, *Relating the bounded arithmetic and polynomial-time hierarchies.* To appear, *Annals of Pure and Applied Logic*, 199?
3. B. Dreben and S. Aanderaa, *Herbrand analyzing functions*, Bulletin of the American Mathematical Society, 70 (1964), pp. 697–698.
4. B. Dreben, P. Andrews, and S. Aanderaa, *False lemmas in Herbrand*, Bulletin of the American Mathematical Society, 69 (1963), pp. 699–706.
5. B. Dreben and J. Denton, *A supplement to Herbrand*, Journal of Symbolic Logic, 31 (1966), pp. 393–398.
6. W. D. Goldfarb, *Herbrand's theorem and the incompleteness of arithmetic*, Iyyun, A Jerusalem Philosophical Quarterly, 39 (1990), pp. 45–64.
7. ———, *Herbrand's error and Gödel's correction*, Modern Logic, 3 (1993), pp. 103–118.
8. J. Herbrand, *Recherches sur la théorie de la démonstration*, PhD thesis, University of Paris, 1930.

[6] Herbrand [9, p.552]

9. ——, *Investigations in proof theory: The properties of true propositions*, in From Frege to Gödel: A Source Book in Mathematical Logic, 1978-1931, J. van Heijenoort, ed., Harvard University Press, Cambridge, Massachusetts, 1967, pp. 525–581. Translation of chapter 5 of [8], with commentary and notes, by J. van Heijenoort and B. Dreben.

10. ——, *Écrits logique*, Presses Universitaires de France, Paris, 1968. Ed. by J. van Heijenoort.

11. ——, *Logical Writings*, D. Reidel, Dordrecht-Holland, 1971. Ed. by W. Goldfarb, Translation of [10].

12. J. Krajíček, P. Pudlák, and G. Takeuti, *Bounded arithmetic and the polynomial hierarchy*, Annals of Pure and Applied Logic, 52 (1991), pp. 143–153.

13. G. Kreisel, *On the interpretation of non-finitist proofs–part I*, Journal of Symbolic Logic, 16 (1951), pp. 241–267.

14. ——, *On the interpretation of non-finitist proofs, part II. interpretation of number theory, applications*, Journal of Symbolic Logic, 17 (1952), pp. 43–58.

15. M. S. Paterson and M. N. Wegman, *Linear unification*, J. Comput. System Sci., 16 (1978), pp. 158–167.

16. J. A. Robinson, *A machine-oriented logic based on the resolution principle*, J. Assoc. Comput. Mach., 12 (1965), pp. 23–41.

17. G. Takeuti, *Proof Theory*, North-Holland, Amsterdam, 2nd ed., 1987.

18. D. Zambella, *Notes on polynomially bounded arithmetic.* To appear in *J. Symb. Logic.*

Some Consequences of Cryptographical Conjectures for S^1_2 and EF

Jan Krajíček and Pavel Pudlák

Mathematical Institute of the Academy of Sciences
Žitná 25, Praha 1, 115 67, Czech Republic

Abstract. We show that there is a pair of disjoint $\mathcal{NP}$-sets, whose disjointness is provable in S^1_2 and which cannot be separated by a set in $\mathcal{P}/poly$, if the cryptosystem RSA is secure. Further we show that factoring and the discrete logarithm are implicitly definable in any extension of S^1_2 admitting an $\mathcal{NP}$ -definition of primes about which it can prove that no number satisfying the definition is composite.
As a corollary we obtain that the Extended Frege (EF) proof system does not admit a feasible interpolation theorem unless the RSA cryptosystem is not secure, and that an extension of EF by tautologies τ_p (p primes), formalizing that p is not composite, as additional axioms does not admit feasible interpolation theorem unless factoring and the discrete logarithm are in $\mathcal{P}/poly$.

The aim of this note is to present a pair of disjoint $\mathcal{NP}$ -sets, whose disjointness is provable in S^1_2 and which cannot be separated by a set in $\mathcal{P}/poly$ if the cryptosystem RSA is secure, and to implicitly define factoring and the discrete logarithm in a natural $\forall\Pi^b_1$-extension of S^1_2 . Such examples are interesting, since they set some limitations on possible independence proofs in bounded arithmetic and lower bound proofs in propositional calculus. In particular we will answer a question of [7] about lower bounds for the interpolation theorem, see Corollary 8, and a question of [14] about such $\mathcal{NP}$ -pairs, see Corollary 9. Furthermore we will discuss the unprovability of the pigeonhole principle in S^1_2.

As this is intended to be a short technical note, we shall omit definition of most of the concepts. For definitions of RSA and one-way functions see e.g. [10], a reference for propositional logic and bounded arithmetic is [6]. The symbol (a, b) denotes the greatest common divisor. Δ^b_1-formulas are everywhere in the paper meant as Δ^b_1 w.r.t. S^1_2.

1 A disjoint $\mathcal{NP}$ -pair based on RSA

The pair is essentially the well known probabilistic encryption schema of [1], which is known to be as secure as RSA, the most famous public-key cryptosystem [15]. We only have to define it suitably, so that the disjointness is provable in S^1_2. For $i = 0, 1$, let

$$A_i =_{df} \{(n, e, y);\ \exists x, d, r < n\ (x \equiv i \bmod 2 \wedge x^e \equiv y \bmod n$$

$$\wedge\, y^d \equiv x \bmod n \wedge y^r \equiv 1 \bmod n \wedge (e,r) = 1\} \ .$$

One can check that exponentiation modulo a number $n \geq 2$ is definable and satisfies the usual relations in S^1_2, namely $x^{y+z} \equiv x^y x^z \bmod n$, $x^{yz} \equiv (x^y)^z \bmod n$. Assuming that RSA is secure against an adversary computing functions in $\mathcal{P}/poly$, this pair cannot be separated by a set in $\mathcal{P}/poly$ even if one considers only those n's which are products of two primes.

Theorem 1.

$$S^1_2 \vdash A_0 \cap A_1 = \emptyset.$$

In particular, assuming that RSA is secure against an adversary computing functions in $\mathcal{P}/poly$, not all Δ^b_1-implications provable in S^1_2 admit an interpolant in $\mathcal{P}/poly$.

Proof.

We shall show in S^1_2 that for every y there is at most one $x < n$ which satisfies the defining formula of either A_0 or A_1. The proof is easy, since we put everything in the definition. So suppose that for some $x_0, x_1 < n$, r_0 (we do not need r_1), d_0, d_1, $y^{r_0} \equiv 1 \bmod n$, $(e, r_0) = 1$ and

$$x_i^e \equiv y \bmod n \wedge y^{d_i} \equiv x_i \bmod n,$$

for $i = 0, 1$. We have

$$x_i^{r_0} \equiv (y^{d_i})^{r_0} \equiv (y^{r_0})^{d_i} \equiv 1 \bmod n,$$

for $i = 0, 1$. Using Euclid's algorithm we can show that there exists an inverse d' to e modulo r_0. Thus, using $x_i^{r_0} \equiv 1 \bmod n$, we have for $i = 0, 1$,

$$y^{d'} \equiv x_i^{ed'} \equiv x_i \bmod n,$$

whence $x_0 = x_1$.

The statement $A_0 \cap A_1 = \emptyset$ can be written as the following implication

$$\alpha_0(n, e, y, x_0, d_0, r_0) \rightarrow \neg\alpha_1(n, e, y, x_1, d_1, r_1)$$

where α_i are Δ^b_1-formulas such that the Σ^b_1-formulas $\exists x, d, r < n\; \alpha_i(n, e, y, x, d, r)$ define A_i, for $i = 0, 1$. Any interpolant of this implication separates A_0 from A_1. This yields the second part of the theorem. □

The advantage of this pair is that we do not have to mention primes as is the case with other examples based on conjectured one-way functions. The problem with primes is that it is unlikely that the $\mathcal{NP}$-definition of primes of Pratt [12] is provably in S^1_2 equivalent to the natural $co\mathcal{NP}$-definition. If it were so, we would immediately get a polynomial time algorithm for primality from Buss's theorem [2]. (It is unlikely that we can get it in this way.) More precisely, we would need only to prove that every number is either a (natural) composite number or a "Pratt prime".

The definition of the sets A_i we use is by no means unique and various modifications are possible. Razborov suggested to replace $y^d \equiv x \bmod n \wedge y^r \equiv 1 \bmod n$ by $x^r \equiv 1 \bmod n$. S.Buss pointed out that by posing an extra condition $(y,n) = 1$ in the definition we may drop r altogether (note that $y^r \equiv 1 (\text{mod } n)$ implies that y, n are coprime). That is because we can define then $r := d \cdot e - 1$ for which

$$y^{r+1} \equiv y^{de} \equiv x^e \equiv y \ (\text{mod } n)$$

This implies, using the new condition $(y,n) = 1$, that

$$y^r \equiv 1 \ (\text{mod } n)$$

and the proof of Theorem 1 continues as before, using $d' := d$.

However, some condition implying $\exists r (y^r \equiv 1 \bmod n)$ seems to be needed, as the next theorem shows.

Theorem 2. *If RSA is secure, then*

$$S_2^1 \nvdash \forall n > 1 \forall y, \ (y,n) = 1 \rightarrow \exists r > 0 \ (y^r \equiv 1 \bmod n) \ .$$

Proof.

Suppose the converse. Then, again by Buss's theorem, we get an r by a polynomial time algorithm. Such an r suffices to break the RSA. Namely, if $y \equiv x^e \bmod n$ and $(e, r') = 1$, where r' is the order of y, then $r'|r$ and hence also $r'|r_0$ for $r_0 = r/(e,r)$. Thus we have also $(e, r_0) = 1$ and $y^{r_0} \equiv 1 \bmod n$, and we compute the inverse of e modulo r_0 and continue as in the proof of Theorem 1. □

In particular, if RSA is secure then S_2^1 cannot prove the Fermat-Euler theorem. Surely it is possible to get a lot of independence results using such assumptions, however the above sentence is rather special, since it easily follows from the weak Pigeon Hole Principle (WPHP). Thus we get:

Corollary 3. *If RSA is secure, then* $S_2^1 \nvdash WPHP(\Delta_1^b)$.

Proof.

Fix n and y prime to each other and take the function $r \mapsto y^r \bmod n$. By $WPHP$ we get $r_1 < r_2 < 2n$ such that $y^{r_1} \equiv y^{r_2} \not\equiv 0 \bmod n$, whence $y^{r_2 - r_1} \equiv 1 \bmod n$. □

Note, in particular, that as S_2 proves $WPHP(\Delta_1^b)$ (by [11]) this gives another proof of the conditional separation $S_2^1 \neq S_2$ based on a different structural complexity conjecture than the proof of [9]. Moreover, it follows from [9, 4] that $PV \nvdash WPHP(\Delta_1^b)$ and $S_2^1 \nvdash PHP(\Delta_1^b)$ (assuming that $NP \not\subseteq P/poly$ and $L^{NP} \neq \Delta_2^p$ respectively) but not the conclusion of Corollary 3.

2 Implicit definability of factoring and the discrete logarithm

Assume that S^1_2 proves

$$A(x,y) \wedge A(x',y) \to x = x'$$

for some Σ^b_1-formula A. In other words, it implicitly defines a (partial) function $x := f(y)$ such that

$$\exists x A(x,y) \to A(f(y),y) \ .$$

We show that in a natural extension of S^1_2 factoring and the discrete logarithm are implicitly definable in this way.

Let $C(p,w)$ be a Δ^b_1-formula saying that

$$w = (g, p, q_1, e_1, \dots, q_t, e_t, w_1, \dots, w_t)$$

such that:

1. $g \in Z^*_p$ and $g^{p-1} \equiv 1 \bmod p$
2. $p - 1 = \Pi_{i \leq t} q_i^{e_i}$
3. $g^{\frac{p-1}{q_i}} \not\equiv 1 \bmod p$, for all $i \leq t$
4. $C(q_i, w_i)$, for all $i \leq t$.

The $\mathcal{NP}$ -definition of primes $Pratt(a)$ by Pratt [12] is $\exists w \leq t(a) C(a,w)$ for a suitable term $t(a)$. Denote by Φ the following $\forall \Pi^b_1$-formula:

$$\Phi := \forall x, Pratt(x) \to \neg Composite(x)$$

where $Composite(a) := \exists u, v < a, u \cdot v = a$.

Theorem 4. *The theory* $S^1_2 + \Phi$ *implicitly defines factoring, i.e. it proves the sequent:*

$$(p, q, p', q' \in Pratt), p \leq q, p' \leq q', p \cdot q = a, p' \cdot q' = a \longrightarrow p = p' \wedge \ q = q' \ .$$

Proof.
We first show in $S^1_2 + \Phi$ the following

Claim: $(Pratt(p) \wedge p|ab) \ \to \ (p|a \vee p|b)$.

Assume $Pratt(p)$ and $p|ab$ but that p does not divide a. Then (a,p) (which is definable in S^1_2) must be equal to 1 by Φ and we get (also in S^1_2)

$$pu + av = 1$$

for some u, v, and so also:

$$pub + avb = b \ .$$

By $p|ab$, p divides the left-hand side and hence $p|b$ as well. This proves the claim.

To prove the theorem let

$$pq = p'q' \ .$$

Then $p|p'q'$ and by the claim $p|p'$ or $p|q'$. But, by Φ, this implies that $p = p'$ or $p = q'$. The same holds for q and, by the assumption $p \leq q$ and $p' \leq q'$, we get $p = p'$ and $q = q'$.

□

Corollary 5. *Unless factoring is in $\mathcal{P}/poly$, not all Δ_1^b-implications provable in $S_2^1 + \Phi$ admit an interpolant in $\mathcal{P}/poly$.*

Proof.

Recall that $Pratt(a)$ is the Σ_1^b-formula $\exists w \leq t(a), C(a,w)$. Denote by $D(a,w)$ the Δ_1^b-formula $(w \leq t(a) \wedge C(a,w))$ and by $(a)_i$ the Δ_1^b-definable function computing the i^{th} bit of a. By Theorem 4 all Δ_1^b-implications

$$D(p,u), D(q,v), p \leq q, p \cdot q = a, (p)_i = 1 \longrightarrow$$

$$\longrightarrow \neg D(p',u'), \neg D(q',v'), p' > q', p' \cdot q' \neq a, (p')_i = 1$$

(and similarly with $(q)_i$ and $(q')_i$ in place of $(p)_i$ and $(p')_i$) are provable in $S_2^1 + \Phi$. Interpolants of such implications compute bits $(p)_i$ and $(q)_i$ from a, hence one of them is not in $\mathcal{P}/poly$ unless factoring itself is in $\mathcal{P}/poly$.

□

Now we want a similar statement for the discrete logarithm.

Theorem 6. *The theory $S_2^1 + \Phi$ implicitly defines the discrete logarithm, i.e. it proves the sequent:*

$$C(p,w), y \equiv g^x \bmod p, y \equiv g^{x'} \bmod p, x < p, x' < p \longrightarrow x = x' \ .$$

Proof.

Assume $x - x' = r > 0$. As $g^x \equiv g^{x'} \bmod p$ we have $g^r \equiv 1 \bmod p$. By $C(p,w)$ also $g^{p-1} \equiv 1 \bmod p$ and hence also $g^s \equiv 1 \bmod p$, where $s = (r, p-1)$. Note that $s < p - 1$ as $r < p - 1$.

For all $i \leq t$ we have $(q_i^{e_i}, \frac{p-1}{q_i^{e_i}}) = 1$; we could put this condition into $C(p,w)$ but it follows from the claim in the proof of Theorem 4 and the following corollary of Φ:

Claim: $Pratt(q) \wedge a|q^u \rightarrow a = q^v$, some $v \leq u$.

If $a|q^u$ then $ab = q^u$ for some b and hence $q|ab$. By the claim from Theorem 4, $q|a$ or $q|b$. Iterating this u-times gives $a = q^v$, $b = q^{v'}$, for some v, v'.

Using a simple property of gcd available in S_2^1 , namely:

$$(a,u) \cdot (a,v) = (a, u \cdot v) \text{ for } (u,v) = 1$$

we get

$$s = \Pi_{i \leq t}(r, q_i^{e_i}) \ .$$

We want to show that, for some j, $(r, q_j^{e_j}) < q_j^{e_j}$ holds. Assume, for the sake of contradiction, that $(r, q_i^{e_i}) = q_i^{e_i}$ for all $i \leq t$. Then by the above

$$s = \Pi_{i \leq t} q_i^{e_i} = p - 1$$

which contradicts to $r < p - 1$. So let $j \leq t$ be a fixed j such that $(r, q_j^{e_j}) < q_j^{e_j}$. Then $(r, q_j^{e_j}) = q_j^{f_j}$, for some $f_j < e_j$. Here we use the above claim again. Thus we get

$$s = (\ \Pi_{i \leq t, i \neq j}(r, q_i^{e_i}) \ ) \cdot q_j^{f_j}$$

and so

$$s \cdot [\ (\ \Pi_{i \leq t, i \neq j} \frac{q_i^{e_i}}{(r, q_i^{e_i})} \ ) \cdot q_j^{e_j - f_j - 1} \ ] = \frac{p-1}{q_j}$$

as $e_j - f_j - 1 \geq 0$. That is:

$$s | \frac{p-1}{q_j} \ .$$

This yields

$$q^{\frac{p-1}{q_j}} \equiv q^{s \cdot u} \equiv 1^u \equiv 1 \bmod p$$

where u is the bracket $[\ldots]$ in the equation above.

But this contradicts the condition from $C(p, w)$:

$$q^{\frac{p-1}{q_j}} \not\equiv 1 \bmod p \ .$$

□

Similarly to Corollary 5 we get

Corollary 7. *Unless the discrete logarithm is in $\mathcal{P}/poly$, not all Δ_1^b-implications provable in $S_2^1 + \Phi$ admit an interpolant in $\mathcal{P}/poly$.*

3 Bounds to interpolation theorem for *EF*

We shall discuss the relation of the above example to a question on interpolation theorems for propositional proof systems studied in [7], and a question of [14] on separating certain pairs of $\mathcal{NP}$ -sets associated with propositional proof systems.

Let us rephrase the first question. Let a propositional proof system P be given. The problem is to determine the complexity of the function $F(d, x)$ which for a proof d of a sequent $\Phi(\bar{p}, \bar{q}) \rightarrow \Psi(\bar{p}, \bar{r})$ and a truth assignment x for $\bar{p}$

- gives 1, if $\exists y \Phi(x, y)$ and
- gives 0, if $\exists z \neg \Psi(x, z)$.

The relation to interpolation theorems is the following. If we fix the sequent and the proof, then

$$\Phi(x,y) \to (F(d,x)=1) \to \Psi(x,z).$$

Thus $F(d,\bar{p})=1$ is like an interpolant. If, in particular, F is in $\mathcal{NC}_1/poly$, then we get an interpolation formula whose size is polynomial in the size of d; if F is in $\mathcal{P}/poly$, then we get an interpolation circuit of polynomial size.

Razborov [14] proposed to study the following pair of disjoint $\mathcal{NP}$-sets for a proof system P:

$$SAT^* =_{df} \{(\Theta, 1^t); \Theta \text{ is a satisfiable CNF}\},$$

$$REF(P) =_{df} \{(\Theta, 1^t); \Theta \text{ is a CNF and } \neg\Theta \text{ has a } P\text{-proof of length } t\}.$$

(1^t is just a padding of length t.) The problem is: what can be the complexity of a separating set?

We observe that these problems are essentially equivalent. In fact, their equivalence as well as the completeness result for this pair follow simply from known relations between bounded arithmetic and propositional logic (see [7]). For the reader not familiar with those relations we sketch a direct argument.

Let us agree that we shall omit the restriction of being CNF in the definition of SAT^* and $REF(P)$ and, instead, let us take all formulas in some complete basis.

First suppose that we can effectively separate SAT^* and $REF(P)$. We shall show how to interpolate a sequent $\Phi(\bar{p},\bar{q}) \to \Psi(\bar{p},\bar{r})$ with a proof d of length t. Let an assignment a for $\bar{p}$ be given. Consider the formula $\neg\Psi(a,\bar{r})$. If it is satisfiable, then our procedure for separating SAT^* and $REF(P)$ will tell us that and we put $F(d,a)=0$. If, on the other hand, $\Phi(a,\bar{q})$ is satisfiable, then we get a proof of $\Psi(a,\bar{r})$ at most polynomially longer than d, since we only need to evaluate $\Phi(a,\bar{q})$ on the satisfying assignment and then apply the proof of the sequent. Thus our procedure will tell us the answer also in this case.

For the converse we have to assume that the proof system P is sufficiently strong, namely, that it proves instances of its own Reflection Principle by proofs of polynomial size. This is true for EF and its various extensions; if stated carefully, it can be proved also for some (apparently) weaker systems, see [6].

So assume that we can effectively compute an interpolation function $F(d,x)$. Let a formula Θ of length n be given. Consider the reflection principle for $\neg\Theta$:

$$Prf_P^{t,n}(\ulcorner\neg\Theta\urcorner, \bar{q}) \to \neg\Theta,$$

where the propositional variables $\bar{q}$ encode proofs of length t and $\ulcorner\neg\Theta\urcorner$ is a truth assignment which encodes $\neg\Theta$. Thus the sets of variables of the formulas in the sequent are disjoint, hence F depends only on the proof d of this sequent, which is of polynomial size in t and n. If Θ is satisfiable, then the negation of the consequent is satisfiable, hence $F(d)=0$; if $\neg\Theta$ has a proof of size $\leq t$, then the antecedent is satisfiable, hence $F(d) = 1$. Thus we can effectively separate SAT^* and $REF(P)$.

Let us note, that the above argument shows that the most difficult sequent for P, from the point of view of interpolation, is

$$Prf_P^{t,n}(\bar{p},\bar{q}) \to SatNeg^n(\bar{p},\bar{r}),$$

where $SatNeg^n(\bar{p},\bar{r})$ expresses that the negation of the formula coded by $\bar{p}$ is satisfied on the truth assignment coded by $\bar{r}$.

The interest in this problem stems from the fact that an upper bound for interpolation for P can yield a lower bound to the size of P-proofs. The idea, discussed in [5, Sec.5] and implemented for independence results for bounded arithmetic in [13] and for lower bounds for propositional logic in [7], is to prove an upper bound on the complexity of an interpolation function F for a proof system P, and then find a pair of disjoint $\mathcal{NP}$ -sets, which are harder to separate than it is to compute F. The natural encoding of the disjointness condition for the pair thus gives a sequent which cannot have a short P-proof.

On the other hand, if such a sequent does have a short P-proof then one gets a lower bound for the interpolation function for P. In this way Theorem 1 gives:

Corollary 8. *Assuming that RSA is secure against an adversary computing functions in $\mathcal{P}/poly$, no interpolation function for EF is in $\mathcal{P}/poly$, i.e., there is a sequence of sequents which have polynomial size EF-proofs, but do not have polynomial size interpolation circuits.*

Proof.
Let $\exists x,d,r\ \alpha_i(n,e,y,x,d,r)$ be the Σ_1^b-formulas defining A_i, $i = 0,1$, from Section 1. Since S_2^1 proves

$$\exists x,d,r\ \alpha_0(n,e,y,x,d,r) \to \neg\exists x,d,r\ \alpha_1(n,e,y,x,d,r),$$

and it is a Π_1^b-formula, the translations of this formula into propositional calculus have polynomial size EF-proofs (see [3, 8] or [6, Chpt.9]). If RSA is secure, we cannot interpolate them by polynomial size circuits. □

Note that in [5] it was shown that constant depth Frege systems do not admit $\mathcal{NC}^1$-bound for interpolation unless $\mathcal{P}/poly \subseteq \mathcal{NC}^1$. The next corollary answers a question of [14].

Corollary 9. *Assuming that RSA is secure against an adversary computing functions in a class C closed under polynomial time reductions, the pair of $\mathcal{NP}$ -sets $(SAT^*, REF(EF))$ cannot be separated by a set in C.*

Proof.

This follows from the discussion above, or using Razborov's completeness result [14] as follows. Represent the sets A_i in V_1^1 using sets encoding the binary representations of numbers. Then, by RSUV-isomorphism [16], we get from Proposition 1:

$$V_1^1 \vdash A_0 \cap A_1 = \emptyset.$$

Since $(SAT^*, REF(EF))$ is complete for such provably disjoint $\mathcal{NP}$-sets, (A_0, A_1) can be reduced to it and we get the corollary. □

4 A problem about EF

Propositions of section 2 show that it is quite important to determine whether the formula Φ is provable in S_2^1. In fact, for those propositions it would be sufficient to have any $\mathcal{NP}$-definition of primes (in N) about which S_2^1 can prove that no such prime is composite. (The condition $C(p,w)$ in the sequent in Theorem 6 would be then replaced by the conjunction of the first three conditions of $C(p,w)$ together with $\bigwedge_{i \le t} w_i \le t(q_i) \wedge D(q_i, w_i)$, where $D \in \Delta_1^b$ and $\exists w \le t(a) D(a, w)$ is such an $\mathcal{NP}$-definition of primes.) The only obstacle to a proof of Φ in S_2^1 is that we are unable to prove in S_2^1 a corollary of Fermat-Euler theorem:

$$n > 1 \wedge (y,n) = 1 \;\rightarrow\; \exists r < n, y^r \equiv 1 \bmod n \; .$$

To see this assume that $p \in Pratt$ with g the primitive root of p from a witness w of the fact that $p \in Pratt$. For the sake of contradiction assume that $p \in Composite$. Then p is either a product $p = ab$ of two coprime a, b or p is a power $p = a^\ell$ of some $a \notin Composite$ and $\ell \geq 2$ (this is obtained in S_2^1 by looking at a decomposition $p = a \cdot b$ with a of the minimal possible length, employing the claim from the proof of Theorem 6). The corollary of Fermat-Euler theorem would then imply that the orders of g modulo a and b are $r < a$ and $s < b$ respectively. Hence the order of g modulo p is at most $rs \le (a-1)(b-1) < p-1$ in the case $p = ab$, and at most $ra^{\ell-1} \le (a-1)a^{\ell-1} < p-1$ in the case $p = a^\ell$. This contradicts the assumption that g is a primitive root of p.

Note that by Theorem 2 it is unlikely that the above corollary of Fermat-Euler theorem is provable in S_2^1.

The following simple theorem shows that the existence of such a formula is equivalent to a polynomial upper bound on the lenghts of EF proofs of primality.

Theorem 10. *The following two propositions are equivalent:*

1. *There is a Σ_1^b-formula $A(a)$ such that:*
   (a) $\{p \in \mathrm{N} \mid A(p)\}$ is the set of primes.
   (b) $S_2^1 \vdash A(a) \rightarrow \neg Composite(a)$.
2. *The tautologies τ_p:*

$$\tau_p := \|\neg Composite(a)\|^n(\bar{p})$$

(a natural propositional translation of the Π_1^b-sentence $\neg Composite(p)$) have polynomial size EF-proofs for all primes p.

Proof.

If 2. holds then the Σ_1^b-formula formalizing that τ_p has an EF-proof of size $\leq poly(|p|)$, suitable *poly*, satisfies condition 1., as EF is provably sound in S_2^1 .

On the other hand, having A satisfying 1. construct an EF-proof of τ_p as follows. Take a witness to $A(p)$ and prove that $A(p)$ holds, and combine this by modus ponens with a polynomial size EF-proof of the translations of $A(p) \rightarrow \neg Composite(p)$, getting just τ_p. For details of the translations see [8] or [6].

□

Hence we have a very interesting problem:

Problem: *Do the tautologies τ_p admit polynomial size EF-proofs?*

We do not know the answer even for constant-depth systems.

Acknowledgement

We thank S.Buss, S.Cook and A.A.Razborov for their comments on the preliminary version of this paper.

References

1. W.B. Alexi, B.Chor, O. Goldreich, C.P. Schnorr (1988) *RSA and Rabin functions: Certain parts are as hard as the whole,* SIAM J. Comp., 17, pp.194-209.
2. S.R. Buss (1986) *Bounded Arithmetic,* Bibliopolis.
3. Cook, S. A. (1975) *Feasibly constructive proofs and the propositional calculus,* in: *Proc. 7th Annual ACM Symp. on Theory of Computing*, pp. 83-97. ACM Press.
4. Krajíček, J. (1993) *Fragments of bounded arithmetic and bounded query classes,* Transactions of the A.M.S., **338(2)** : 587-598.
5. ——— (1994) *Lower bounds to the size of constant-depth propositional proofs,* Journal of Symbolic Logic, **59(1)**:73-86.
6. ——— (1994) *Bounded arithmetic, propositional logic and complexity theory,* Cambridge University Press, in print.
7. ——— (1994) *Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic,* submitted.
8. Krajíček, J., and Pudlák, P. (1989) *Propositional proof systems, the consistency of first order theories and the complexity of computations,* J. Symbolic Logic, **54(3)**:1063-1079
9. Krajíček, J, Pudlák, P, and Takeuti, G. (1991) *Bounded arithmetic and the polynomial hierarchy,* Annals of Pure and Applied Logic, **52**: 143–153.
10. Papadimitriou, A. (1994) *Computational complexity,* Addison-Wesley.
11. Paris, J, and Wilkie, A. (1985) *Counting problems in bounded arithmetic*, in: Methods in Mathematical Logic, LNM 1130, pp.317-340.Springer.
12. Pratt, V.R. (1975) *Every prime has a succinct certificate*, SIAM J. Computing, 4:214-220.
13. A.A. Razborov (1994) *Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic*, Izvestiya of the RAN, to appear.

14. ——— (1994) *On provably disjoint NP-pairs,* Basic Research in Computer Science Center, Aarhus, RS-94-36, preprint.

15. M. Rivest, A. Shamir and L. Adleman (1978) *A method of obtaining digital signatures and public-key cryptosystems.* ACM Communications 21, pp. 120-126.

16. G. Takeuti (1992) *RSUV isomorphism,* in Arithmetic, Proof Theory and Computational Complexity, Clote and Krajíček eds., Oxford Univ. Press, pp. 364-386.

Frege Proof System and TNC°

Gaisi Takeuti*

Frege proof system F is any usual system of propositional calculus e.g. a Hilbert style system based on finitely many axiom schemes and inference rules. Extended Frege system EF is obtained from F as follows. An EF-sequence is a sequence of formulas $\psi_1, \dots, \psi_k$ such that each ψ_i is either an axiom of F, inferred from previous ψ_u and $\psi_v (= \psi_u \to \psi_i)$ by modes pones or of the form $q \leftrightarrow \varphi$, where q is an atom occurring neither in φ nor in any of $\psi_1, \dots, \psi_{i-1}$. Such $q \leftrightarrow \varphi$, is called an extension axiom and q a new extension atom. An EF-proof is any EF-sequence whose last formula does not contain any extension atom. In [12], S. A. Cook and R. Reckhow proved that the pigeonhole principle PHP has a simple polynomial size EF-proof and conjectured that PHP does not admit polynomial size F-proof. In [4], S. R. Buss refuted this conjecture by furnishing polynomial size F-proof for PHP. Since then the important separation problem of polynomial size F and polynomial size EF has not had any progress.

In [11], S. A. Cook introduced the system PV, a free variable equational logic whose provable functional equalities are 'polynomial time verifiable' and showed that metamathematics of F and EF can be developed in PV and the soundness of EF is proved in PV. In [2], S. R. Buss introduced the first order system S_2^1 and showed that S_2^1 is essentially conservative extension of PV. There he also introduced a second order system $V_1^1(BD)$. In [23] we proved that S_2^1 and $V_1^1(BD)$ are isomorphic under so-called RSUV Isomorphism. In [16], J. Krajíček proved that a proof in $V_1^1(BD)$ more precisely $\Sigma_1^{1,b}$-part of $V_1^1(BD)$ is simulated in polynomial size EF-proof and therefore that a proof in S_2^1 (more precisely Σ_1^b-part of S_2^1) can be simulated by a polynomial size EF-proof.

In Clote-Takeuti [10], we introduced a first order system TNC° which corresponds to computational complexity class NC^1. We also introduced another first order system $T^\circ NC^\circ$ which is equivalent to $T^\circ NC^\circ$.

In this paper we first develop metamathematics of F and EF in TNC° and prove the soundness of F in TNC°. Then we show that a proof in $T^\circ NC^\circ$ is simulated in polynomial size F proof. We actually prove a stronger statement.

Let n_0 be the number of propositional variables in an F-proof or an EF-proof. Let f be an NC^1-function defined in TNC° and TNC° proves that $f(a)$ is an F-proof (or an EF-proof) of its conclusion $g(a)$ where $a = 2^{n_0}$. If we substitute 1, 2, 3,... for n_0 in $f(a)$, then $f(2^1)$, $f(2^2)$, $f(2^3), \dots$ express F-proofs (or EF-proofs) P_1, P_2, $P_3, \dots$. In this case we say "the sequence P_1, P_2, $P_3, \dots$ is a

[*] Partially supported by the grant #93025 of the joint U.S.-Czechoslovak Science and Technology Program

uniform F-proof (or a uniform EF-proof). More precisely $f(a)$ itself is called a uniform F-proof (or a uniform EF-proof). Then we have the following theorem.

Theorem. *If P is a proof of A in $T^\circ NC^\circ$, then there exists a uniform F-proof of $\ulcorner \tilde{A} \urcorner$ which is a natural interpretation of A.*

If P is a proof of A in S^1_2 where A is an NC^1 formula expressed as a sharply bounded formula in $T^\circ NC^\circ$, then there exists a uniform EF-proof of $\ulcorner \tilde{A} \urcorner$ which is a natural interpretation of A.

Since the equivalence of an NC^1-formula A and natural evaluation of natural interpretation of A is provable in TNC°, we have the following corollary.

Corollary. *If* TNC° *and S^1_2 are separated by an NC^1-formula, then the uniform F and the uniform EF are separated.*

At the end we discuss examples in computatonal complexity namely Razborov's clique, Raz-Wigderson's matching and Karchmer-Wigderson's *st*-connectivity from our point of view. These examples do not belong to monotone NC^1. Therefore it is hard to describe them in Frege system. Nevertheless their works used sort of implicit definitions of them which are stated in Freg system. We show that the statements used in the implicit definitions are polynomial size F provable. This might suggest the possibility that S^1_2 is conservative over TNC° with respect to the NC^1-formulas.

Then we prove that the statement in Karchmer-Wigdenson's example has a constant depth polynomial size F proof and also polynomial size cut-free LK proof with substitution but has no polynomial size cut-free LK proof.

§1. TNC° and $T^\circ NC^\circ$.

In this section, we discuss the properties of TNC° and $T^\circ NC^\circ$ which we will use in the next section.

For the presentation of TNC° and $T^\circ NC^\circ$, we first introduce TAC° and $T^\circ AC^\circ$ both of which correspond to AC°.

The language of TAC° and $T^\circ AC^\circ$ consists of 0, 1, +, $2^{|y|} \cdot x$, $\dot{-}$, $|x|$, $x\#y$, $\lfloor \frac{1}{2}x \rfloor$, MSP, and $\leq$, where the intended meaning of $x\#y$ is $2^{|x|\cdot|y|}$ and the defining equations of MSP are

$$MSP(a,0) = 0 \quad \text{and}$$
$$MSP(a,i+1) = \left\lfloor \frac{1}{2} MSP(a,i) \right\rfloor .$$

Now let T be a theory in Bounded Arithmetic. A formula is said to be *esb* (essentially sharply bounded) in T if it belongs to the smallest family $\mathcal{F}$ satisfying the following conditions. See [10] for the detail.

(1) Every atomic formula belongs to $\mathcal{F}$.

(2) $\mathcal{F}$ is closed under Boolean connectives.

(3) $\mathcal{F}$ is closed under sharply bounded quantifications.

(4) If $A(\vec{a}, x)$ and $B(\vec{a}, x)$ belong to $\mathcal{F}$ and

$T \vdash \exists x \leq s(\vec{a}) A(\vec{a}, x)$ and

$$T \vdash c \leq s(\vec{a}), d \leq s(\vec{a}), A(\vec{a}, c), A(\vec{a}, d) \rightarrow c = d$$

where c and d do not occur in $s(\vec{a})$ and $A(\vec{a}, x)$, then $\exists x \leq s(\vec{a})(A(\vec{a}, x) \wedge B(\vec{a}, x))$ and $\forall x \leq s(\vec{a})(A(\vec{a}, x) \supset B(\vec{a}, x)$ belong to $\mathcal{F}$.

The theory TAC° is formulated by the defining axioms of the basic functions and predicates of the language and the following axioms and inferences.

(1) Bit-Extensionality Axioms.

$$|a| = |b|, \quad \forall i < |a|(\mathrm{Bit}(i, a) = \mathrm{Bit}(i, b)) \rightarrow a = b$$

where $2 \cdot a = a + a$, $\mathrm{mod2}(a) = a \dot{-} 2 \cdot \lfloor \frac{1}{2} a \rfloor$, and
$\mathrm{Bit}(i, a) = \mathrm{mod2}(MSP(a, i))$.

(2) Bit-comprehension Axioms

$$\exists y < 2^{|s|} \forall i < |s|(\mathrm{Bit}(i, y) = 1 \leftrightarrow A(i)),$$

where $A(i)$ is *esb*.

(3) *esb*-LIND

$displaystyle\frac{A(a),\Gamma\rightarrow\Delta,A(a+1)}{A(0),\Gamma\rightarrow\Delta,A(|t|)}$ where $A(a)$ is *esb* and a satisfies the eigenvariable condition i.e. a does not occur in the lower sequent.

Let T be a theory in bounded Arithmetic and $A(\vec{a}, b)$ be *esb* in T. Suppose

$$T \vdash \forall \vec{x} \exists y \leq t(\vec{x}) A(\vec{x}, y),$$
$$T \vdash \forall \vec{x} \forall y \forall z (A(\vec{x}, y) \wedge A(\vec{x}, z) \supset y = z)$$

and $\forall \vec{x} A(\vec{x}, f(\vec{x}))$ is satisfied. Then T is said to *esb*-defines the function f.

The following theorem was proved in [10].

Theorem 1. *A predicate is in* AC° *iff it is expressed by an esb formula in* TAC°. *A function is in* AC° *iff it is esb-definable in* TAC°.

In [10], a weak theory of short sequences was developed in TAC°. Especially the following concepts and functions are expressed by *esb* formulas or *esb*-definable functions in TAC°.

$\mathrm{Seq}(w)$: w is a sequence of natural numbers.
$\beta(i, w)$: i-th component of $w(i > 0)$.
$\mathrm{right}(w)$: $|\mathrm{right}(w)|$ is a bound for $|\beta(i, w)|$.
$\mathrm{Len}(w)$: the length of w.
$\mathrm{Sq\,Bd}(b, d)$: an upperbound of the sequence numbers with the length $\leq |d|$ whose components have the length $\leq |b|$.

Also $\mu x \leq |t| A(x)$ i.e. the minimum $x \leq |t|$ satisfying $A(x)$ is *esb*-definable if $A(x)$ is *esb*.

Now the theory $T°AC°$ is obtained from TAC° by replacing *esb* formulas by sharply bounded formulas. Then the following theorem was proved in [10].

Theorem 2. *Every esb formula in* TAC° *is equivalent to a sharply bounded formula in* TAC° *and* TAC° *and* $T°AC°$ *are equivalent.*

The language of TNC° is the same with the language of TAC°. TNC° is obtained from TAC° by introducing the following *esb*-BSN (bounded successive nomination)

$$\frac{b \le k \to \exists! y \le k A(i,b,y)}{\begin{aligned} &s \le k \to \exists w \le \mathrm{Sq\,Bd}(k,t)(\mathrm{Seq}(w) \wedge \mathrm{right}(w) = k \\ &\qquad \wedge \mathrm{Len}(w) = |t| \wedge \beta(1,w) = s \\ &\qquad \wedge \forall i < |t| A(i,\beta(i+1,w),\beta(i+2,w)),\end{aligned}}$$

where b is an eigenvariable, k is a numeral, and $A(i,b,y)$ is *esb* in TNC°.

Theorem 3. *A predicate is in* NC^1 *iff it is expressed by an esb formula in* TNC°. *A function is in* NC^1 *iff it is esb-definable in* TNC°

See [10] for a proof.

The sum of bits of a is denoted by Count and has the following defining axioms.

$$\mathrm{Count}(0) = 0$$

$$a > 0 \to \mathrm{Count}(a) = \mathrm{Count}(a - 2^{|a| \dot{-} 1}) + 1.$$

Count is in NC^1 and expressed by an *esb* formula in TNC°.

A term of the form $|t|$ is said to be small and a term of the form $||t||$ is said to be very small.

In [10], we also introduced a theory $T°NC°$ which is equivalent to TNC°. Here we use a slightly different formulation of $T°NC°$ such that the equivalence of two versions is obvious.

The language of $T°NC°$ is the language of TAC° together with $\mathrm{or}(x,y)$, $\mathrm{and}(x,y)$, $\mathrm{orand}(x_0,x_1,x_2,x_3)$, and $\mathrm{orand}(x,i)$.

$T°NC°$ is obtained from $T°AC°$ by introducing the following defining axioms on $\mathrm{or}(x,y)$, $\mathrm{and}(x,y)$, $\mathrm{orand}(x_0,x_1,x_2,x_3)$, and $\mathrm{orand}(x,i)$.

$$\mathrm{or}(x_0,x_1) = \begin{cases} 0 & \text{if both } x_0 \text{ and } x_1 \text{ are } 0 \\ 1 & \text{otherwise}. \end{cases}$$

$$\mathrm{and}(x_0,x_1) = \begin{cases} 1 & \text{if both } x_0 \text{ and } x_1 \text{ are } > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{or\,and}(x_0,x_1,x_2,x_3) = \mathrm{or}(\mathrm{and}(x_0,x_1),\mathrm{and}(x_2,x_3))$$

$$\mathrm{or\,and}(x,0) = \mathrm{or\,and}(\mathrm{Bit}(0,x),\mathrm{Bit}(1,x),\mathrm{Bit}(2,x),\mathrm{Bit}(3,x))$$

For a very small i,

$$\begin{aligned} \mathrm{or\,and}(x,i+1) = {} & \mathrm{or\,and}(\mathrm{or\,and}(y_0,i),\mathrm{or\,and}(y_1,i), \\ & \mathrm{or\,and}(y_2,i),\mathrm{or\,and}(y_3,i)) \end{aligned}$$

where

$$\begin{aligned} y_0 &= LSP(x,4(i+1)), \\ y_1 &= LSP(MSP(x,4(i+1)),4(i+1)), \\ y_2 &= LSP(MSP(x,8(i+1)),4(i+1)), \\ y_3 &= LSP(MSP(x,12(i+1)),4(i+1)), \end{aligned}$$

and LSP satisfies the following axiom

$$a = MSP(a,i) \cdot 2^i + LSP(a,i).$$

The intuitive meaning of y_0, y_1, y_2, y_3 is expressed by $x = y_0 + y_1 \cdot (16)^{i+1} + y_2(16)^{2(i+1)} + y_3(16)^{3(i+1)} + \cdots$.

Theorem 4. *A predicate is in* NC^1 *iff it is expressed by a sharply bounded formula in* $T^\circ NC^\circ$. *A function is in* NC^1 *iff it is definable by a sharply bounded formula in* $T^\circ NC^\circ$. $T^\circ NC^\circ$ *and* TNC$^\circ$ *are equivalent.*

See [10] for a proof.

¿From now on by an NC^1 formula we mean either a sharply bounded formula in $T^\circ NC^\circ$ or an *esb* formula in TNC$^\circ$.

Similarly by an NC^1 function we mean a function definable in a sharply bounded formula in $T^\circ NC^\circ$ or an *esb*-definable function in TNC$^\circ$.

Let $A(x)$ be *esb* in TNC$^\circ$. Then there exists a unique y such that

$$y < 2^{|t|+1} \wedge \forall i \leq |t|(\mathrm{Bit}(i,y) = 1 \leftrightarrow A(i)).$$

We define $\#x \leq tA(x)$ by Count(y).

Obviously $\#x \leq t(A(x))$ is *esb*-definable in TNC$^\circ$.

§2. Metamathematics of F and EF.

In this section we develop metamathematics of F and EF in TNC$^\circ$.

We define $i \in a$ to be $\mathrm{Bit}(i,a) = 1$. Then the following comprehension axiom holds.

$$\exists x < 2^n \forall i < n(i \in x \leftrightarrow A(i)),$$

where n is small and $A(i)$ is an *esb* formula without any occurrence of x.

Our convention nicely make a notion of finite set of natural numbers but it has a conflict with a more commonly used convention to identify n and $\{0, 1, \ldots, n-1\}$ since we have to represent $\{0, 1, \ldots, n-1\}$ by $2^n \dot{-} 1$.

Therefore we have to be very careful when we mention a function. We say "f is a function defined on n" if $f(0)$, $f(1), \ldots, f(n \dot{-} 1)$ are defined. In this case we can identify a function defined on n and an *esb*-definable sequence w whose length n and for every $i < n$, the $i+1$ component $\beta(i+1, w)$ of w is equal to $f(i)$.

Definition. A function f is the enumeration map of a iff f is defined on Count(a) onto a satisfying

$$\forall i, j < \mathrm{Count}(a)(i < j \leftrightarrow f(i) < f(j)).$$

Proposition 5. *The enumeration map of a is in* NC^1 *and its existence and uniqueness can be proved in* TNC°.

Proof. Let $n = |a|$. First we prove by induction on i

$$\forall i < \mathrm{Count}(a) \exists j < |a| (\mathrm{Count}(LSP(a, j+1)) = i + 1).$$

For $i = 0$, we can take j to be

$$\mu j < |a| (\mathrm{Bit}(a, j) = 1).$$

Suppose $i = i_0 + 1$ and $\mathrm{Count}(LSP(a, j_0 + 1)) = i_0 + 1$. Then we can take j to be

$$\mu j < n(j > j_0 \wedge \mathrm{Bit}(a, j) = 1).$$

Now we can define $f(i)$ by the following equation

$$f(i) = \mu j < n\mathrm{Count}(LSP(a, j+1)) = i + 1. \qquad \Box$$

Definition. A function $g : a \to b$ is defined to be the pair of f and h such that f is the enumeration function of a and h is a function defined on n into b, where n is $\mathrm{Count}(a)$. In usual mathematical notation

$$g(i) = h(f^{-1}(i)).$$

Definition. A nonempty set t is said to be a tree if

(1) $0 \notin t$,

(2) $\forall v \in t(|v| > 1 \supset \lfloor v/2 \rfloor \in t)$.

For a tree t, one can show $1 \in t$. 1 is called the root of t. Let v_1, $v_2 \in t$. We say 'v_1 is the predecessor of v_2' or 'v_2 is a successor of v_1' if $\lfloor v_2/2 \rfloor = v_1$. Every v in a tree except the root has a unique predecessor and every v in a tree has at most two successors. If v has a successor v', then v' must be either $2v$ or $2v+1$.

Let v_1 and v_2 be in a tree t. We say 'v_1 is an ancestor of v_2 in t' denoted by $v_1 \overset{t}{<} v_2$ iff $\exists i < |v_2|(v_1 = MSP(v_2, i)) \wedge v_1 \neq v_2$. $\overset{t}{<}$ is a partial order on $'$. We say 'v_2 is a descendant of v_1 in t' if v_1 is an ancestor of v_2 in t. If v is in t and $v \neq 1$, then 1 is ancestor of v.

If v has a successor, v is called a node. If v has no successor, v is called a leaf. For every node there exists a leaf such that the node is an ancestor of the leaf.

Definition. A tree t is said to be left weighted if for every node v of t, $2v$ is always a successor of v. So if t is left weighted and a node v in t has a unique successor, the successor must be $2v$.

For the purpose of arithmetization of metamathematics, we use 2 for $\neg$, 3 for $\to$, 4 for $\vee$, and 5 for $\wedge$. We fix a small number n_0 and $6, 7, \ldots, 5+n_0$ represent propositional variables and $6+n_0, \ldots, 5+2n_0$ represent the negation of variables represented by 6, 7, $\ldots, 5+n_0$ respectively. We denote 6, 7, $\ldots, 5+n_0$ by p_0, $p_1, \ldots, p_{n_0-1}$ and $6+n_0, \ldots, 5+2n_0$ by $\bar{p}_0$, $\bar{p}_1, \ldots, \bar{p}_{n_0-1}$.

A formula is a pair (t, f) where t is a left-weighted tree and

$$f : t \to \{0, 1, \ldots, 5+2n_0\}$$

satisfying the following conditions.

1. If v is a node in t, then $f(v)$ is one of $\neg$, $\rightarrow$, $\vee$ and $\wedge$. If $f(v)$ is one of $\rightarrow$, $\vee$, and $\wedge$, then v has two successors. If $f(v)$ is $\neg$, then v has only one successor.
2. If v is a leaf in t, then $f(v)$ is one of 0, 1, $p_0, \ldots, p_{n_0-1}$, $\bar{p}_0, \ldots, \bar{p}_{n_0-1}$.

Let t be a tree and $v \in t$. Then $|v| \dot{-} 1$ is called the height of v. The height of t is defined to be the maximum height of its elements. The height h of t is very small and satisfies $|t| = 2^{h+1}$.

A formula (t, f) is said to be a $(\vee, \wedge)$-formula if for every node v in t, $f(v)$ is either $\vee$ or $\wedge$.

A $(\vee, \wedge)$-formula (t, f) is said to be a normal $(\vee, \wedge)$-formula if the following condition is satisfied.

For every node with an even height in t, $f(v) = \vee$ and for every node with an odd height, $f(v) = \wedge$.

A normal $(\vee, \wedge)$-formula (t, f) is said to be complete if the following condition is satisfied.

The height h of t is even and t is $\{1, 2, \ldots, 2^{h+1} \dot{-} 1\}$. I.e. every v with $2^h \leq v < 2^{h+1}$ is a leaf in t and every v with $1 \leq v < 2^h$ is a node in t.

A transformation of a $(\vee, \wedge)$-formula into a normal $(\vee, \wedge)$-formula.

Let $v > 0$ have a height h. For $i \leq h$, v_i is defined by the following equation

$$v_i = MSP(v, h-i).$$

Let (t, f) be a $(\vee, \wedge)$-formula, $v \in t$, and the height of v be h. We define "$\tilde{v}$ corresponds to v" by the following conditions.

1. The height of $\tilde{v}$ is $2h$.
2. If $i < h$ and $f(v_i) = \vee$ and $v_{i+1} = 2v_i$ i.e. v_{i+1} is the left successor of v_i, then $\tilde{v}_{2(i+1)}$ is $4\tilde{v}_{2i}$ or $4\tilde{v}_{2i} + 1$ i.e. $\tilde{v}_{2(i+1)}$ is a successor of the left successor of $\tilde{v}_{2i}$
3. If $i < h$ and $f(v_i) = \wedge$ and $v_{i+1} = 2v_i + 1$ i.e. v_{i+1} is the right successor of v_i, then $\tilde{v}_{2(i+1)}$ is $4\tilde{v}_{2i} + 2$ or $4\tilde{v}_{2i} + 3$ i.e. $\tilde{v}_{2(i+1)}$ is a successor of the right successor of $\tilde{v}_{2i}$.
4. If $i < h$ and $f(v_i) = \wedge$ and $v_{i+1} = 2v_i$ i.e. v_{i+1} is the left successor of v_i, then $\tilde{v}_{2(i+1)}$ is $4\tilde{v}_{2i}$ or $4\tilde{v}_{2i}+2$ i.e. the left successor of a successor of $\tilde{v}_{2i}$.
5. If $i < h$ and $f(v_i) = \wedge$ and $v_{i+1} = 2v_i + 1$ i.e. v_{i+1} is the right successor of v_i, then $\tilde{v}_{2(i+1)}$ is $4\tilde{v}_{2i} + 1$ or $4\tilde{v}_{2i} + 3$ i.e. the right successor of a successor of $\tilde{v}_{2i}$.

Let us denote "$\tilde{v}$ corresponds to v" by $A(v, \tilde{v})$. Then $A(v, \tilde{v})$ is a sharply bounded formula. Then by induction on the height of v $h(v)$ we can show $\forall v \in t \exists \tilde{v} A(\tilde{v})$.

Since the height of t is very small, the height of $\tilde{v}$ is very small and therefore $\tilde{v}$ is small. Therefore we can define $\# \tilde{v} A(v, \tilde{v})$ and show $\# \tilde{v} A(v, \tilde{v}) = 2^{h(v)}$ by the induction on $h(v)$. We define $v' \dot{<} v$ by $\exists i < h(v)(v' = v_i)$. Then we have

$$v' \dot{<} v \to \tilde{v}' \dot{<} \tilde{v} \quad \text{and}$$
$$v' \neq v \to \tilde{v}' \neq \tilde{v}$$

by induction on $\max(h(v), h(v'))$ where we assume $A(v', \tilde{v}')$ and $A(v, \tilde{v})$.

We define $\tilde{t}$ to be the set of all $\tilde{v}'$ which satisfies $\tilde{v}' = \tilde{v}$ or $\tilde{v}' \dot{<} \tilde{v}$ for some $\tilde{v}$ which satisfies $A(v, \tilde{v})$ for some leaf v in t. We can also prove that for every $\tilde{v}$ with even height $2h$ in $\tilde{t}$ there exists a unique v with the height h in t such that $\tilde{v}$ corresponds to v and that $A(v, \tilde{v})$ implies that v is a leaf in t iff $\tilde{v}$ is a leaf in $\tilde{t}$.

Now we can define a normal $(\vee, \wedge)$-tree$(\tilde{t}, \tilde{f})$ by defining $\tilde{f}(\tilde{v})$ for a leaf $\tilde{v}$ by $\exists v(\tilde{f}(\tilde{v}) = f(v) \wedge A(v, \tilde{v}))$.

One can easily see that the intended logical meaning of (t, f) is logically equivalent to the intended logical meaning of $(\tilde{t}, \tilde{f})$.

A transformation of a normal $(\vee, \wedge)$-formula into a complete normal $(\vee, \wedge)$-formula.

Let (t, f) be a $(\vee, \wedge)$-normal tree with a height $2h$. Then define t^* to be

$$\{1, 2, \ldots, 2^{2h+1} - 1\}$$

and a complete normal $(\vee, \wedge)$-formula (t^*, f^*) is defined by the following condition.

If v^* is a leaf of t^* and v is the unique leaf in t satisfying $v \dot{<} v^* \vee v = v^*$, then

$$f^*(v^*) = f(v).$$

It is easily seen that intended logical meaning of (t, f) is logically equivalent to the intended logical meaning of (t^*, f^*).

A transformation of a normal $(\neg, \to)$-formula into a $(\vee, \wedge)$-formula.

A formula (t, f) is said to be $(\neg, \to)$-formula if for every node v in t, $f(v)$ is either $\neg$ or $\to$. A $(\neg, \to)$-formula (t, f) is said to be normal, if for every leaf v in t, $f(v)$ is one of 0, 1, p_0, $p_1, \ldots, p_{n_0 - 1}$.

Let (t, f) be a normal $(\neg, \to)$-formula. It t does not have any node with two successors, then (t, f) must be of the form $\neg \cdots \neg p$ where p is 0, 1, or p_i. Let h be the height of t. Then h is the number of the negations in (t, f). If h is even, then we transform (t, f) to (t_0, f_0), where t_0 consists solely of the root i.e. 1 and $f_0(1) = p$. If h is odd, then we transform (t, f) to (t_0, f_0) where t_0 is the same as before and $f_0(1)$ is $\bar{p}$ i.e. $f_0(1)$ is 1, 0, or $\bar{p}_i$ respectively if $f(1)$ is 0, 1, or p_i.

Now let t have a node with two successors and $\bar{v}_0$ be the topmost node with two successors. Then for every $v \in t$, v is an ancestor of $\bar{v}_0$, $v = \bar{v}_0$, or v is a descendant of $\bar{v}_0$, since otherwise v and $\bar{v}_0$ must meet at some same ancestor v' and v' must have two successor.

Let t^* be the subset of t which consists of all nodes with two successors and all leaves. Let v and v' be two members of t^*. We say "v' is the left t^*-successor of v" if v' is $2v$ or the smallest member of t^* which is a descendant of $2v$ in t. We also say "v' is the right t^*-successor of v", if v' is $2v+1$ or the smallest member of t^* which is a descendant of $2v+1$ in t.

We say "v is a t^*-leaf" if v is a leaf in t. We say "v is a t^*-node" if v is not a t^*-leaf. It is easily seen that every t^*-node v has two t^*-successors. We define $f^* : t^* \to \{\vee, \wedge\} \cup \{0, 1, p_0, \ldots, p_{n_0-1}, \bar{p}_0, \ldots, \bar{p}_{n_0-1}\}$ as follows.

Let $v \in t^*$ and h be the height of v. If v is a t^*-node and $\#i < |v|(\mathrm{Bit}(i, v) = 0)$ is even (or odd), then $f^*(v) = \vee$ (or $f^*(v) = \wedge$). If v is a t^*-leaf, $f(v) = p$ and $\#i < |v|(\mathrm{Bit}(i, v) = 0)$ if even (or odd), then $f^*(v) = p$ (or $f^*(v) = \bar{p}$). Here we make a convention that $\bar{p}$ is 1, 0, $\bar{p}_i$ respectively if p is 0, 1, p_i.

Now let $v \in t^*$. We define "$\tilde{v}$ corresponds to v" as follows.

1. If v is the topmost element $\bar{v}_0$, then $\tilde{v}$ is 1.
2. If v is the left t^*-successor of v' and $\tilde{v}'$ corresponds to v', then $\tilde{v}$ is $2\tilde{v}'$.
3. If v is the right t^*-successor of v' and $\tilde{v}'$ corresponds to v', then $\tilde{v}$ is $2\tilde{v}' + 1$.

We denote "$\tilde{v}$ corresponds to v" by $A(v, \tilde{v})$ for a while. Then it is easily seen that there exists a unique $\tilde{v}$ which corresponds to v We define $\tilde{t}$ to be the set of $\tilde{v}$ which corresponds to some member of t^*. We define $\tilde{f} : \tilde{t} \to \{0, 1, \vee, \wedge, p_0, \ldots, p_{n_0-1}, \bar{p}_0, \ldots, \bar{p}_{n_0-1}\}$ by

$$\tilde{f}(\tilde{v}) = f^*(v)$$

where v is the unique element of t^* to which $\tilde{v}$ corresponds. It is easily seen that the intended logical meaning of $(\tilde{t}, \tilde{f})$ is logically equivalent to the intended logical meaning of (t, f).

A transformation of a normal $(\neg, \vee)$-formula into a $(\neg, \to)$-formula.

A formula (t, f) is said to be $(\neg, \vee)$-formula if for every node v in t, $f(v)$ is either $\neg$ or $\vee$. A $(\neg, \vee)$-formula (t, f) is said to be normal if for every leaf v in t, $f(v)$ is one of 0, 1, $p_0, \ldots, p_{n_0-1}$.

Let (t, f) be a normal $(\neg, \vee)$-formula and $v \in t$. We say "$\tilde{v}$ corresponds to v" if the following conditions are satisfied.

1. If v is 1, then $\tilde{v}$ is 1.
2. If v' is the predecessor of v, v' has two successors, $v = 2v'$ (or $2v' + 1$), and $\tilde{v}'$ corresponds to v', then $\tilde{v}$ is $4\tilde{v}'$ (or $2\tilde{v}' + 1$).
3. If v' is the predecessor of v, v' has a unique successor $v = 2v'$ and $\tilde{v}'$ corresponds to v', then $\tilde{v} = 2\tilde{v}'$.

It is easily seen that for every $v \in t$ there exists a unique $\tilde{v}$ which corresponds to v.

Let $\tilde{t}$ be the set of all $\tilde{v}_i$, where $\tilde{v}_i$ is the i-th subsection of $\tilde{v}$ and $\tilde{v}$ corresponds to some $v \in t$.

We define $\tilde{f}$ unique by the condition that $(\tilde{t}, \tilde{f})$ is a $(\neg, \rightarrow)$-formula and $\tilde{f}(\tilde{v}) = f(v)$ if $\tilde{v}$ is a leaf in $\tilde{t}$ and $\tilde{v}$ corresponds to v. It is easily seen that the intended logical meaning of $(\tilde{t}, \tilde{f})$ is logically equivalent to the intended logical meaning of (t, f).

A transformation of a normal $(\vee, \wedge)$-formula to a normal $(\neg, \vee)$-formula. Let (t, f) be a normal $(\vee, \wedge)$-formula and $v \in t$. We say "$\tilde{v}$ corresponds to v" if the following conditions are satisfied.

1. If v is 1, then $\tilde{v}$ is 1.
2. If v is the left successor (or the right successor) of v', $f(v') = \vee$, $f(v) = \vee$, and $\tilde{v}'$ corresponds to v', then $\tilde{v}$ is the left successor i.e. $2\tilde{v}'$ (or the right successor i.e. $2\tilde{v}' + 1$) of $\tilde{v}'$.
3. If v is the left successor (or the right successor) of v', $f(v') = \vee$, $f(v) = \wedge$, and $\tilde{v}'$ corresponds to v', then $\tilde{v}$ is $4\tilde{v}'$ (or $2(2\tilde{v}' + 1)$).
4. If v is the left successor (or the right successor) of v', $f(v') = \wedge$, $f(v) = \wedge$, and $\tilde{v}'$ corresponds to v', then $\tilde{v}$ is the left successor i.e. $2\tilde{v}'$ (or the right successor i.e. $2\tilde{v}' + 1$) of $\tilde{v}'$.
5. If v is the left successor (or the right successor) of v', $f(v') = \wedge$, $f(v) = \vee$, and $\tilde{v}'$ corresponds to v', then $\tilde{v}'$ is $4\tilde{v}'$ (or $2(2\tilde{v}' + 1)$).

It is easily seen that for every v in t there exists a unique $\tilde{v}$ which corresponds to v. We define $\tilde{t}$ to be the set of all subsections of $\tilde{v}$ which corresponds to some leaf in t. We define $\tilde{f}$ by the condition that $(\tilde{t}, \tilde{f})$ is a normal $(\neg, \vee)$-formula and $\tilde{f}(\tilde{v}) = f(v)$ if $\tilde{v}$ is a leaf of $\tilde{t}$ and $\tilde{v}$ corresponds to v. It is easily seen that the intended logical meaning of (t, f) is logically equivalent to the intended logical meaning of (t, f).

Definition. Let v and v' have the height h and h' respectively. We define $\tilde{v} = v \star v'$ by the following conditions.

1. The height of $\tilde{v}$ is $h + h'$.
2. $\forall i \leq h(\mathrm{Bit}(h' + i, \tilde{v}) = \mathrm{Bit}(i, v))$
   $\wedge \forall i < h'(\mathrm{Bit}(i, \tilde{v}) = \mathrm{Bit}(i, v'))$

If the binary representations of v and v' are $1\epsilon_1 \cdots \epsilon_h$ and $1\epsilon'_1 \cdots \epsilon'_{h'}$, then the binary representation of $v * v'$ is $1\epsilon_1 \cdots \epsilon_h\ \epsilon'_1 \cdots \epsilon'_{h'}$.

Definition. We denote a formula (t, f) by $\ulcorner A \urcorner$. Our intention of this notation is that (t, f) is a code for a formula A. Let $\circ$ be one of $\vee$, $\wedge$ and $\rightarrow$ and (t_1, f_1) and (t_2, f_2) be $\ulcorner A_1 \urcorner$ and $\ulcorner A_2 \urcorner$ respectively. We define $\ulcorner A_1 \circ A_2 \urcorner$ to be (t, f) defined by the following conditions.

1. t consists of all numbers of the form 1, $2 * v_1$, or $3 * v_2$, where $v_1 \in t_1$ and $v_2 \in t_2$.
2. f is defined by the following equations

$$f(1) = \circ, \quad f(2 * v_1) = f_1(v_1) \quad \text{and} \quad f(3 * v_2) = f_2(v_2).$$

Let (t_0, f_0) be $\ulcorner A \urcorner$. We define $\ulcorner \neg A \urcorner$ to be (t,f) defined by the following conditions.

1. t consists of all numbers of the form 1, $2 * v$ where $v \in t_0$.
2. f is defined by the following equations

$$f(1) = \neg \quad \text{and} \quad f(2 * v) = f_0(v).$$

When we are dealing with normal $(\vee, \wedge)$-formulas A, B or normal $(\neg, \rightarrow)$-formulas A, B, by $\ulcorner A \circ B \urcorner$ (or $\ulcorner \neg A \urcorner$ we mean normal $(\vee, \wedge)$-formulas or $(\neg, \rightarrow)$-formulas transformed from $\ulcorner A \circ B \urcorner$.

Let $(t, f) = \ulcorner A(p_{i_1}, \ldots, p_{i_k}) \urcorner$ be a formula, where k is small but may not be a numeral, and g be an *esb*-definable function in TNC° satisfying that $g(p_{i_j})$ is a formula $(t_j, f_j) = \ulcorner B_j \urcorner$. Then $(\tilde{t}, \tilde{f}) = \ulcorner A(B_1, \ldots, B_k) \urcorner$ is defined by the following conditions.

1. Every leaf in $\tilde{t}$ is either a leaf v of t satisfying the condition that $f(v)$ is none of $p_{i_1}, \ldots, p_{i_k}$, or of the form $v * v'$ where v is a leaf of t with $f(v) = p_{i_j}$ and v' is a leaf in t_j.
2. If $v \in t$ is not a leaf satisfying the condition that $f(v)$ is one of $p_{i_1}, \ldots, p_{i_k}$, then $\tilde{f}(v) = f(v)$.
3. If $v \in t$ is a leaf satisfying $f(v) = p_{i_j}$, then $\tilde{f}(v) = f_j(1)$.
4. If $\tilde{v} = v * v'$ and $f(v) = p_{i_j}$ and $v' \in t_j$, then $\tilde{f}(\tilde{v}) = f_j(v')$.

Definition. Let g be an *esb*-definable function in TNC° such that $g(i) = (t_i, f_i) = \ulcorner B_i \urcorner$ for $i < k$ where k is small. Let h be the smallest number satisfying $k + 1 \leq 2^h$. Then $\ulcorner \bigvee_{i \leq k} B_i \urcorner$ is defined by the following $(\tilde{t}, \tilde{f})$.

1. $\ulcorner A(p_0, \ldots, p_{k-1}) \urcorner = (t, f)$ is defined as follows.
   1.1. $t = \{1, 2, \ldots, 2^{h+1} \dot{-} 1\}$.
   1.2. If v is not a leaf of t, then $f(v) = \vee$.
   1.3. If v is $2^h + i$ with $i \leq k$, then $f(v) = p_i$.
   1.4. If v is $2^h + i$ with $k + 1 \leq i$, then $f(v) = 0$.
2. $(\tilde{t}, \tilde{f}) = \ulcorner A(B_0, \ldots, B_k) \urcorner$ is obtained by (t, f) and g in the previous definition. We define $\ulcorner \bigwedge_{i \leq k} B_i \urcorner$ by replacing $\bigvee$, $\vee$, 0 in the definition of $\ulcorner \bigvee_{i \leq k} B_i \urcorner$ by $\bigwedge$, $\wedge$, 1 respectively.

For the sake of definiteness we shall adopt the following Frege system F. The language of F consists of 0, 1, $p_0, \ldots, p_{n_0 - 1}$, $\neg$, $\rightarrow$. Its axioms are all instances of schemes: $A \rightarrow 1$, $0 \rightarrow A$, $A \rightarrow (B \rightarrow A)$, $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$, $(\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B)$, and its only rule of inference is modus ponens: A and $A \rightarrow B$ entail B. We use $A \wedge B$, $A \vee B$ and $A \leftrightarrow B$ as abbreviations of $\neg(A \rightarrow \neg B)$, $\neg A \rightarrow B$ and $(A \rightarrow B) \wedge (B \rightarrow A)$ respectively.

Extended Frege system EF is constructed as follows. An EF-sequence is a sequence of formulas $A_1, \ldots, A_k$ such that each A_i is an axiom of F, inferred from some previous A_u and $A_v (= A_u \rightarrow A_i)$ by modus ponens or of the form

$q \leftrightarrow A$, where q is a propositional variables (i.e. one of $p_0, \ldots, p_{n_0-1}$) occurring neither A nor any of $A_1, \ldots, A_{i-1}$. Such $q \leftrightarrow A$ is called an extension axiom and q a new extension atom. An EF-proof is any EF-sequence whose last formula does not contain any extension atom.

Let w be a sequence of length k. If $A_1, \ldots, A_k$ is an F-proof (or an EF-proof) and $\ulcorner A_1 \urcorner, \ldots, \ulcorner A_k \urcorner$ is $\beta(1, w), \ldots, \beta(k, w)$, in an abuse of language we say "w is an F-proof (or EF-proof)." It is easily seen that these notions are *esb* definable in TNC°.

A formula (t, f) is called a sentence if for every leaf v in t $f(v)$ is either 0 or 1. A formula (t, f) is denoted by $\mathbb{0}$ if $t = \{1\}$ and $f(1) = 0$. A formula (t, f) is denoted by $\mathbf{1}$ if $t = \{1\}$ and $f(1) = 1$.

Definition. A number $a < 2^{n_0}$ is called an atom evaluation. Let (t, f) be a formula of the form
$\ulcorner A(p_0, \ldots, p_{n_0-1}) \urcorner$ and a be an atom evaluation. Then by $S(t, f, a)$ we denote $\ulcorner A(\epsilon_0, \ldots, \epsilon_{n_0-1}) \urcorner$ where ϵ_i is $\mathbb{0}$ if $\text{Bit}(i, a) = 0$ and $\mathbf{1}$ otherwise. $S(t, f, a)$ is a sentence. When (t, f) is $\ulcorner A \urcorner$, we denote $S(t, f, a)$ by $S(\ulcorner A \urcorner, a)$.

Let (t, f) be a complete normal $(\vee, \wedge)$-sentence with height $2h$ i.e. a complete normal $(\vee, \wedge)$-formula with height $2h$ which is a sentence. Then (t, f) is uniquely determined by $f(2^{2h})$, $f(2^{2h} + 1), \ldots, f(2^{2h+1} \dot{-} 1)$. Let $x < 2^n$ be defined by the formula

$$\forall i < 2^{2h} (\text{Bit}(i, x) = f(2^{2h} + i)),$$

where $n = 2^{2h}$. Then (t, f) is uniquely determined by x and x is *esb*-definable from (t, f). We say "x expresses (t, f)."

Let (t, f) be complete normal $(\vee, \wedge)$-sentence with height $2h$ and x express (t, f). We define evaluation of (t, f) denoted by $\text{eval}(t, f)$ as follows

$$\text{eval}(t, f) = \begin{cases} x & \text{if } h = 0 \\ \text{orand}(x, h-1) & \text{otherwise}. \end{cases}$$

If we denote (t, f) by $\ulcorner A \urcorner$, then we denote $\text{eval}(t, f)$ by $\text{eval}(\ulcorner A \urcorner)$. Then it is easily seen that the truth definition defined by this evaluation is *esb*-definable in TNC° and TNC° proves the usual Tarski's condition. For a detailed description, let (t, f) be a complete normal $(\vee, \wedge)$-sentence with height $2h$, denoted by $\ulcorner A \urcorner$, and expressed by x. Then for $h > 0$, $\ulcorner A \urcorner$ is of the form $\ulcorner (A_0 \wedge A_1) \vee (A_2 \wedge A_3) \urcorner$. The numbers x_0, x_1, x_2, x_3 which express $\ulcorner A_0 \urcorner$, $\ulcorner A_1 \urcorner$, $\ulcorner A_2 \urcorner$, $\ulcorner A_3 \urcorner$, can be computed from x by using MSP and LSP. Then

$$\text{eval}(\ulcorner A \urcorner) = (\text{eval}(\ulcorner A_0 \urcorner) \wedge \text{eval}(\ulcorner A_1 \urcorner)) \vee (\text{eval}(\ulcorner A_2 \urcorner) \wedge \text{eval}(\ulcorner A_3 \urcorner)),$$

if we use the rule $1 \wedge 0 = 0$, $1 \vee 0 = 1$, etc. Since we can transform any formula to a complete normal $(\vee, \wedge)$-formula, this gives an evaluation of the sentences which satisfies the usual Tarski's condition. I.e. $\text{eval}(\ulcorner A \rightarrow B \urcorner) = \text{eval}(\ulcorner A \urcorner) \rightarrow \text{eval}(\ulcorner B \urcorner), \text{eval}(\ulcorner \neg A \urcorner) = \neg \text{eval}(\ulcorner A \urcorner)$ etc.

Theorem 6. TNC° *proves that F is a sound proof system i.e.*

$$\mathrm{Atomeval}(a), \quad Prf_F(w,t,f) \to \mathrm{eval}(S(t,f,a)) = 1,$$

where Atomeval(a) *and* $Prf_F(w,t,f)$ *express "a is an atom evaluation" and "w is an F-proof of a formula (t,f)" respectively.*

Proof is easily done by induction on the length of w.

The following theorem was proved by S. A. Cook in [11], though he used PV in the place of S_2^1.

Theorem 7 (Cook). S_2^1 *proves that EF is a sound proof system i.e.*

$$\mathrm{Atomeval}(a), Prf_{EF}(w,t,f,) \to \mathrm{eval}(S(t,f,a)) = 1,$$

where $Prf_{EF}(w,t,f)$ expresses "w is an EF-proof of a formula (t,f)".

Proof. It is easily shown by induction on the length w that there exists an atom evaluation b which coincides with a for the variables in (t,f) and satisfies all extension axioms used in w. Then the theorem is proved by the use of b and Theorem 6.

Remark. In [3], S. R. Buss proved that the Boolean formula value problem is in ALOGTIME i.e. NC^1 and complete for ALOGTIME under deterministic log time reduction i.e. AC° reduction.

Similar contents were proved with somewhat different technical methods in S. R. Buss, S. A. Cook, A. Gupta and V. Ramachandran [7]. A new and much simpler ALOGTIME algorithm for the Boolean formula value problem was given in S. R. Buss [5]. An in [6], S. R. Buss proved that there are polynomial size Frege proofs of the partial consistency statements for Frege systems. Therefore the large part of this section can be considered as redoing these in our setting.

§3. $NU°$ and $N°U°$.

In [23], we have introduced the RSUV isomorphism. The basic idea of the RSUV isomorphism is that bounded second order objects in the bounded second order Bounded Arithmetic correspond to first order objects in the first order Bounded Arithmetic and that first order objects in the second order theory correspond to lengths of objects in the appropriate first order theory. By a bounded second order object we mean a predicate α on the integers $< a$ for some first order object a which is denoted by α^a. To make the correspondence between a bounded second order object α^a and a first order object b, we interpret the truth values of $\alpha^a(i)$ as the bits in the binary representation of b. This makes a first order interpretation of the second order concept and an inverse second order interpretation of the first order concept. By this way, we can find isomorphism between a first order theory and an appropriate second order theory. This isomorphism is called the RSUV isomorphism between two theories. In [23], it is proved among many other things that S_2^1 is isomorphic to $V_1^1(BD)$. In [24] it is also proved among many other things that TAC° is isomorphic to the

$\Delta_0^{1,b}(BD)$-extension of T_1, which is denoted by $\tilde{T}_1$, and that TNC° is isomorphic to $N\tilde{U}_1^{1,0}$ which is denoted by NU° in this paper.

In this section we present $\tilde{T}_1$, $V_1^1(BD)$, NU°, and $N^\circ U^\circ$ which is isomorphic to $T^\circ NC^\circ$.

The language of $\tilde{T}_1$, the $\Delta_0^{1,b}(BD)$-extension of T_1, consists of

$$0, 1, +, \cdot, |x|, \left\lfloor \frac{1}{2}x \right\rfloor, \leq .$$

Notice that it includes the multiplication but excludes #.

$\tilde{T}_1$ is a second order theory and includes second order variables α^t, $\beta^t, \ldots$ in addition of first order variables where t is a first order term. In the second order theory, we use a notion "abstract". An abstract is of the form $\{x\}A(x)$, where $A(a)$ is a formula and x is a bound variable not occurring in $A(a)$. An abstract $\{x\}A(x)$ is used as a substitution instance for a free second order variable α. For example $F(\{x\}A(x))$ is $A(0) \land \forall x(A(x) \supset A(x+1)) \supset A(a)$ if $F(\alpha)$ is $\alpha(0) \land \forall x(\alpha(x) \supset \alpha(x+1)) \supset \alpha(a)$. $\tilde{T}_1$ has initial sequents expressing defining axioms of function constants and predicates constants in the language and the following sequents in addition of the initial sequents of the form $D \to D$

$$\begin{aligned}
&\alpha^t(s) \to s \leq t \\
&s \leq t_1, \quad s \leq t_2, \quad \alpha^{t_1}(s) \to \alpha^{t_2}(s) \\
&s_1 = s_2, \quad \alpha^t(s_1) \to \alpha^t(s_2)\,.
\end{aligned}$$

The inferences of $\tilde{T}_1$ are the following in addition to the usual first order inferences

$$\frac{F(\{x\}(x \leq t \land A(x))), \Gamma \to \Delta}{\forall \varphi^t F(\varphi^t), \Gamma \to \Delta} \quad \text{and} \quad \frac{\Gamma \to \Delta, F(\{x\}(x \leq t \land A(x)))}{\Gamma \to \Delta, \exists \varphi^t F(\varphi^t)}$$

where $A(a)$ is a $\Delta_0^{1,b}$-formula i.e. a formula without any second order quantifiers or any first order unbounded quantifiers

$$\frac{\Gamma \to \Delta, F(\alpha^t)}{\Gamma \to \Delta, \forall \varphi^t(\varphi^t)} \quad \text{and} \quad \frac{F(\alpha^t), \Gamma \to \Delta}{\exists \varphi^t F(\varphi^t), \Gamma \to \Delta}$$

where α satisfies the eigenvariable condition i.e. any second order variables of the form α^s do not occur in the lower sequent $\Delta_0^{1,b}(BD)$-IND

$$\frac{A(a), \Gamma \to \Delta, A(a+1)}{A(0), \Gamma \to \Delta, A(t)}$$

where a satisfies the eigenvariable condition and $A(a)$ is a $\Delta_0^{1,b}(BD)$-formula.

The following theorem was proved in [24].

Theorem 8. TAC° *and* $\tilde{T}_1$ *are isomorphic by the* RSUV *isomorphism.*

$V_1^1(BD)$ is obtained from $\tilde{T}_1$ by adding the following inference

$\Sigma_1^{1,b}(BD) - \text{IND}$

$$\frac{A(a), \Gamma \to \Delta, A(a+1)}{A(0), \Gamma \to \Delta, A(t)}$$

where $A(a)$ is a $\Sigma_1^{1,b}(BD)$-formula. In [23], the following theorem was proved.

Theorem 9. *S^1_2 and $V^1_1(BD)$ are isomorphic by the* RSUV *isomorphism.*

Let T be a bounded domain second order theory in Bounded Arithmetic. I.e. all second order variables in T are of the form α^t. A formula is said to be essentially elementary bounded in T (abbreviated by *eeb* in T or simply by *eeb*) if it belongs to the smallest family $\mathcal{F}$ satisfying the following conditions

(1) Every atomic formula belongs to $\mathcal{F}$.
(2) $\mathcal{F}$ is closed under Boolean connectives.
(3) $\mathcal{F}$ is closed under bounded quantifications.
(4) If $A(\alpha^s)$ and $B(\alpha^s)$ belong to $\mathcal{F}$ and

$$T \vdash \exists\varphi^s A(\varphi^s) \qquad \text{and}$$
$$T \vdash A(\alpha^s), A(\beta^s) \to \forall x \le s(\alpha^s(x) \leftrightarrow \beta^s(x))$$

where α and β do not occur in $A(\gamma^s)$, then $\exists\varphi^s(A(\varphi^s) \wedge B(\varphi^s))$ and $\forall\varphi^s(A(\varphi^s) \supset B(\varphi^s))$ belong to $\mathcal{F}$.

Now we are going to define NU° which was denoted by $N\tilde{U}^{1,0}_1$.

Let k be a natural number. Any number $a \le k$ can be expressed by a sequence of 0, 1 with the length $|k|$ therefore by a predicate $\alpha^{|k|-1}$. By $\alpha \upharpoonright [a, b]$ we denote $\{x\}(x \le b \dot{-} a \wedge \alpha^b(a+x))$. Then any sequence $a_0, a_1, \dots, a_t$ with $\forall_i \le t(a_i \le k)$ can be expressed by

$$\alpha \upharpoonright [0, |k| \dot{-} 1], \quad \alpha \upharpoonright [|k|, 2|k| \dot{-} 1], \dots, \alpha \upharpoonright [t \cdot |k|, (t+1)|k| \dot{-} 1]$$

by choosing an adequate α.

Let f be defined on $k+1$ satisfying $\forall i \le k(f(i) \le k)$. If $s \le k$, then we have a sequence with length t

$$s, f(s), f(f(s)), \dots, f^{t-1}(s)$$

where $f^1(s) = f(s)$ and $f^{n+1}(s) = f(f^n(s))$. Expressing f by a $1-1$ correspondence defined by a formula and s, $f(s), \dots, f^{t-1}(s)$ by

$$\varphi \upharpoonright [0, |k| \dot{-} 1], \dots, \varphi \upharpoonright [(t-1)|k|, t|k| \dot{-} 1]$$

for a small t, we get the following *eeb*-BSN (bounded successive nomination)

$$\frac{a \le k \to \exists! y \le kA(i, a, y)}{\begin{array}{l} s \le k \to \exists\varphi^{|k|t-1}[\varphi \upharpoonright [0, |k| \dot{-} 1] \overset{H}{=} \{x\}(x < |k| \wedge \mathrm{Bit}(x, s) = 1) \\ \wedge \forall i < t\exists y \le k\{\varphi \upharpoonright [i|k|, (i+1)|k| \dot{-} 1] \overset{H}{=} \{x\}(x < |k| \wedge \mathrm{Bit}(x, y) = 1)\} \\ \wedge \forall i < t \dot{-} 1\forall y_1 \le k\forall y_2 \le k\{\varphi \upharpoonright [i|k|, (i+1)|k| \dot{-} 1] \overset{H}{=} \{x\}(x < |k| \wedge \mathrm{Bit}(x, y_1) = 1\} \\ \wedge \varphi \upharpoonright [(i+1)|k|, (i+2)|k| \dot{-} 1] \overset{H}{=} \{x\}(x < |k| \wedge \mathrm{Bit}(x, y_2) = 1) \supset A(i, y_1, y_2)\}] \end{array}}$$

where k is a positive numeral, a satisfies the eigenvariable condition, $A(i,a,b)$ is *eeb*, and

$$\{x\}(x \leq t \wedge A(x)) \overset{H}{\leq} \{x\}(x \leq s \wedge B(x))$$

is defined to be

$$\forall x \leq t(A(x) \supset x \leq s \wedge B(x)) \vee \exists x \leq s[B(x) \wedge > (x \leq t \wedge A(x)) \wedge \{\forall y \leq t(x < y \wedge A(y)) \supset y \leq s \wedge B(y)\}]$$

and $\{x\}(x \leq t \wedge A(x)) \overset{H}{=} \{x\}(x \leq s \wedge B(x))$ is defined to be

$$\{x\}(x \leq t \wedge A(x)) \overset{H}{\leq} \{x\}(x \leq s \wedge B(x)) \wedge \{x\}(x \leq s \wedge B(x)) \overset{H}{\leq} \{x\}(x \leq t \wedge A(x)).$$

Now NU° is obtained from $\tilde{T}_1$ by adding *eeb*-BSN and the following inferences

$$\frac{F(\{x\}(x \leq t \wedge A(x))), \Gamma \rightarrow \Delta}{\forall \varphi^t F(\varphi^t), \Gamma \rightarrow \Delta} \quad \text{and} \quad \frac{\Gamma \rightarrow \Delta, F(\{x\}(x \leq t \wedge A(x)))}{\Gamma \rightarrow \Delta, \exists \varphi^t F(\varphi^t)}$$

where $A(x)$ is *eeb* in NU°

$$\frac{A(a), \Gamma \rightarrow \Delta, A(a+1)}{A(0), \Gamma \rightarrow \Delta, A(t)}$$

where $A(a)$ is *eeb* in NU° and a satisfies the eigenvariable condition.

The following theorem was proved in [24].

Theorem 10. TNC° *and* NU° *are isomorphic by the RSUV isomorphism.*

Now we define $N^\circ U^\circ$. $N^\circ U^\circ$ is obtained from $\tilde{T}_1$ by introducing third order predicate Orand and the following axioms.

(1) Orand$(\alpha^{s(a)}, 0) \leftrightarrow (\alpha^{s(a)}(0) \wedge \alpha^{s(a)}(1)) \vee (\alpha^{s(a)}(2) \wedge \alpha^{s(a)}(3))$.

(2) For small i,

Orand$(\alpha^{s(a)}, i+1) \leftrightarrow (A_0 \wedge A_1) \vee (A_2 \wedge A_3)$

where

A_0 is Orand$(\alpha^{s(a)}, i)$,

A_1 is Orand$(\{x\}(x < 2^{2i} \wedge \alpha^{2(a)}(2^{2i} + x)))$,

A_2 is Orand$(\{x\}(x < 2^{2i} \wedge \alpha^{2(a)}(2 \cdot 2^{2i} + x)))$,

and A_3 is Orand$(\{x\}(x < 2^{2i} \wedge \alpha^{2(a)}(3 \cdot 2^{2i} + x)))$.

It is easily seen that the following theorem follows from [24].

Theorem 11. $T^\circ NC^\circ$ *and* $N^\circ U^\circ$ *are isomorphic by the RSUV isomorphism.*

Moreover the RSUV isomorphism implies that Orand$(\alpha^{s(a)}, i)$ can be constructed as an *eeb* formula in NU° can be considered as an elementary bounded formula in $N^\circ U^\circ$ i.e. a formula without any second order quantifiers or any first order unbounded quantifiers in $N^\circ U^\circ$.

§4. Uniform F and uniform EF.

First we simulate the $p\Sigma_1^b$-part of S_2^1 in uniform EF which is defined in the introduction. A formula in S_2^1 is said to be pure Σ_1^b denoted by $p\Sigma_1^b$ if it is of the form $\exists x_1 \le t_1 \cdots \exists x_n \le t_n A(x_1,\ldots,x_n)$ where $A(x_1,\ldots,x_n)$ is sharply bounded. A sequent $A_1,\ldots,A_m \to B_1,\ldots,B_n$ is said to be a $p\Sigma_1^b$ sequent all $A_1,\ldots,A_m \to B_1,\ldots,B_n$ are $p\Sigma_1^b$. S_2^1 restricted to $p\Sigma_1^b$ sequents is called the $p\Sigma_1^b$-part of S_2^1. The $p\Sigma_1^b$-part of S_2^1 is sufficient to develop the theory of polynomial time computational class P.

We make similar definitions in $V_1^1(BD)$. A formula in $V_1^1(BD)$ is said to be pure $\Sigma_1^{1,b}$ denoted by $p\Sigma_1^{1,b}$ if it is of the form

$$\exists\varphi_1^{s_1}\cdots\exists\varphi_n^{s_n} A(\varphi_1^{s_1},\ldots,\varphi_n^{s_n})$$

where $A(\varphi_1^{s_1},\ldots,\varphi_n^{s_n})$ is $\Delta_0^{1,b}(BD)$ i.e. elementary bounded.

A sequent $A_1,\ldots,A_m \to B_1,\ldots,B_n$ in $V_1^1(BD)$ is said to be $p\Sigma_1^{1,b}$ if all $A_1,\ldots,A_m$, $B_1,\ldots,B_n$ are $p\Sigma_1^{1,b}$. $V_1^1(BD)$ restricted to the $p\Sigma_1^{1,b}$ sequents is called the $p\Sigma_1^{1,b}$ part of $V_1^1(BD)$. The $p\Sigma_1^{1,b}$-part of S_2^1 and the $p\Sigma_1^{1,b}$-part of $V_1^1(BD)$ are obviously isomorphic by the RSUV isomorphism. Therefore we simulate $p\Sigma_1^{1,b}$-part of $V_1^1(BD)$ in uniform EF in the place of simulation of $p\Sigma_1^b$-part of S_2^1.

In [16], J. Krajíček proved that the $p\Sigma_1^{1,b}$-part of $V_1^1(BD)$ has its simulation in polynomial size EF. We are going to show that we can formalize his simulation using coding in TNC° and prove its uniformity.

Let P be a $p\Sigma_1^{1,b}$ proof in $V_1^1(BD)$. For simplicity, we assume without loss of generality that the end sequence of P has only one first order free variable a.

By the devise in pp. 378–9 in [23], we can assume that all second order variables in P have the same bound $s(a)$ where $s(a)$ is a polynomial of a with positive coefficients.

Moreover we may assume without loss of generality that P satisfies the following normal conditions.

(1) All second order free variables are either in the end sequent of P or used as an eigenvariable. No second order free variables in the end sequent of P are used as an eigenvariable.

(2) Free second order variables used as an eigenvariable in different inferences must be different.

(3) Let a second order free variable α be used as an eigenvariable in the inference I in P. Then every occurrence of α in P must be above I.

We enumerate all second order free variables in P, say $\alpha_0,\alpha_1,\ldots,\alpha_{k-1}$, and also some second order bound variables, say $\varphi_0,\varphi_1,\ldots,\varphi_{\ell(a)-1}$, where $\ell(a)$ is a polynomial of a. We define $n_0 = k(s(a)+1)$ and $\tilde{n}_0 = n_0 + \ell(a)(s(a)+1)$.

We let n_0 play the role of n_0 in §2 but for a while we use $\tilde{n}_0$ in the place of n_0. More precisely we denote $\alpha_i^{s(a)}(j)$ by $\tilde{p}_{i(s(a)+1)+j}$ for $j \le s(a)$ and $\varphi_i^{s(a)}(j)$ by $\tilde{p}_{2n_0+i(s(a)+1)+j}$.

Now let A be an eb (elementary bounded) formula in which all second order free variables are among $\alpha_0^{s(a)},\ldots,\alpha_{k-1}^{s(a)}$.

We define $\langle A\rangle$ by using formalization of metamathematics of EF. But we use NU° version in the place of TNC° version. Therefore n_0 is not small now and *esb* will be replaced by *eeb* now.

$\langle A\rangle$ is a formalized formula denoted by $\ulcorner\tilde{A}\urcorner$ where $\ulcorner\tilde{A}\urcorner$ is a code of propositional formula $\tilde{A}$.

$\langle A\rangle$ is defined in NU° as follows.

(1) If $A(a_1,\ldots,a_r)$ is an atomic formula without any of $\alpha_0,\ldots,a_{k-1}$, then $\langle A(a_1,\ldots,a_r)\rangle$ is

$$\mu x\leq 1(A(a_1,\ldots,a_r)\wedge x=1).$$

Certainly $\langle A(a_1,\ldots,a_r)\rangle$ is *eeb*-definable in NU°. $\langle A(a_1,\ldots,a_r)\rangle$ is either 0 or 1 and of the form $\ulcorner\tilde{A}\urcorner$.

(2) If A is of the form $\alpha_i^{s(a)}(t(a_1,\ldots,a_n))$, then $\langle A\rangle$ is

$$\mu x\leq 5+n_0(t(a_1,\ldots,a_n)\leq s(a)\wedge x=p^{\alpha_i}_{t(a_1,\ldots,a_n)})$$

where $p^{\alpha_i}_{t(a_1,\ldots,a_n)}$ is $5+i(s(a)+1)+t(a_1,\ldots,a_n)$. Then $\langle A\rangle$ is either 0 or some p_r for $r\leq 5+n_0$. Therefore it is of the form $\ulcorner\tilde{A}\urcorner$.

(3) If A is of the form $\neg B$, then $\langle A\rangle$ i.e. $\ulcorner\tilde{A}\urcorner$ is $\ulcorner\neg\tilde{B}\urcorner$

(4) If A is of the form $B\to C$, then $\langle A\rangle$ i.e. $\ulcorner\tilde{A}\urcorner$ is $\ulcorner\tilde{B}\to\tilde{C}\urcorner$.

(5) If A is of the form $\forall x\leq t(\vec{a})B(x,\vec{a})$ (or $\exists x\leq t(\vec{a})B(x,\vec{a})$), then $\langle A\rangle$ i.e. $\ulcorner\tilde{A}\urcorner$ is $\ulcorner\bigwedge\limits_{i\leq t(\vec{a})}\tilde{B}(i,\vec{a})\urcorner$ (or $\ulcorner\bigvee\limits_{i\leq t(\vec{a})}\tilde{B}(i,\vec{a})\urcorner$), where $\ulcorner\tilde{B}(i,\vec{a})\urcorner$ for $t\leq t(\vec{a})$ and $\wedge$-operation (or $\vee$-operation) inside $\ulcorner\quad\urcorner$ is also *eeb*-definable in NU°.

Let A be of the form $B(\alpha_i^{s(a)})$ and $\langle A\rangle$ be $\ulcorner\tilde{A}\urcorner$. Then $\tilde{A}$ is of the form $\tilde{B}(p_0^{\alpha_i},\ldots,p_{s(a)}^{\alpha_i})$. Let $D(x)$ be elementary bounded and $\langle D(x)\rangle$ be $\ulcorner\tilde{D}(x)\urcorner$. Then the simultaneous substitution of $\tilde{D}(0),\ldots,\tilde{D}(s(a))$ for $p_0^{\alpha_i},\ldots,p_{s(a)}^{\alpha_i}$ in $\ulcorner\tilde{B}(p_0^{\alpha_i},\ldots,p_{s(a)}^{\alpha_i})\urcorner$ is *eeb*-definable in NU° and the result is denoted by $\ulcorner\tilde{B}(\bar{D}(0),\ldots,\tilde{D}(s(a)))\urcorner$.

If C is of the form $B(\{x\}(x\leq s(a)\wedge D(x)))$, then we can prove by induction on the number of logical symbols in B that $\langle C\rangle$ i.e. $\ulcorner\tilde{C}\urcorner$ is equivalent to $\ulcorner\tilde{B}(\tilde{D}(0),\ldots,\tilde{D}(s(a)))\urcorner$ in NU°. From now on $\tilde{B}(p_0^{\alpha_i},\ldots,p_{s(a)}^{\alpha_i})$ is denoted by $\tilde{B}(p^{\alpha_i})$ and $\tilde{B}(\tilde{D}(0),\ldots,\tilde{D}(s(a)))$ is denoted by $\ulcorner\tilde{B}(\tilde{D})\urcorner$. Now we define a natural atom evaluation $\{x\}B_0(x)$ in NU° to be

$$\begin{aligned}&\{x\}(x\leq k(s(a)+1)\\&\wedge\exists i<k\exists j\leq s(a)(x=p_{i(s(a)+1)+j}\wedge\alpha_i^{s(a)}(j)))\end{aligned}$$

where $\exists i<kC(i)$ is an abbreviation of $C(0)\vee\ldots\vee C(k-1)$. Since $n_0=k(s(a)+1)$, $\{x\}B_0(x)$ is an atom evaluation in the sense that to each $p_{i(s(a)+1)+j}$ it assigns its true value according to $\alpha_i^{s(a)}(j)$.

In §2, a sentence $S(\ulcorner A\urcorner,a)$ is *esb*-defined in TNC°. By the RSUV isomorphism, a sentence $\tilde{S}(\ulcorner\tilde{A}\urcorner,\{x\}B_0(x))$ is *eeb*-defined in NU°, where all second order free variables in A are among $\alpha_0^{s(a)},\ldots,\alpha_{k-1}^{s(a)}$. In the same way, eval in §2 which is *esb*-definable in TNC° is transformed by the RSUV isomorphism to Eval which is *eeb*-definable in NU°. Then we have the following lemma.

Lemma 12. *Let A be an elementary bounded and all second order free variables in A be among $\alpha_0^{s(a)}, \ldots, \alpha_{k-1}^{s(a)}$. Then NU° proves*

$$A \leftrightarrow \ Eval\ (\tilde{S}(\ulcorner \tilde{A} \urcorner, \{x\}B_0(x)))$$

where $\ulcorner \tilde{A} \urcorner$ is $\langle A \rangle$.

Proof. It is easily proved by the induction on the number of logical symbol in A, since all the properties of eval and S are translated to the properties of Eval and $\tilde{S}$ by the RSUV isomorphism. □

Now let P be a proof in $p\Sigma_1^{1,b}$ part of $V_1^1(BD)$. Without loss of generality we may further assume that every sequent in P is of the form

$$\begin{aligned}&\exists\psi_1^{s(a)} B_1(\vec{a}, \alpha_0^{s(a)}, \psi_1^{s(a)}), \ldots, \exists\psi_u^{s(a)} B_u(\vec{a}, \alpha_0^{s(a)}, \psi_u^{s(a)})\\ &\to \exists\varphi_1^{s(a)} C_1(\vec{a}, \alpha_0^{s(a)}, \varphi_1^{s(a)}), \ldots, \exists\varphi_v^{s(a)} C_v(\vec{a}, \alpha_0^{s(a)}, \varphi_v^{s(a)}),\end{aligned}$$

where $B_1, \ldots, B_u$, $C_1, \ldots, C_v$ are elementary bounded and have no second order variables other than ones indicated there.

Then we prove by induction on the number of inferences in P that there exist $\alpha_i, \ldots, \alpha_{i_u}$ and some new bound variable $\varphi_{i_1}, \ldots, \varphi_{i_v}$ such that

$$\begin{aligned}&\ulcorner \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_{i_1}}) \land \ldots \land \tilde{B}_u(\vec{a}, p^{\alpha_0}, p^{\alpha_{i_u}})\\ &\qquad\qquad\qquad \to \tilde{C}_1(\vec{a}, p^{\alpha_0}, p^{\alpha_{j_1}}) \lor \ldots \lor \tilde{B}_v(\vec{a}, p^{\alpha_0}, p^{\alpha_{i_v}}) \urcorner\end{aligned}$$

is *eeb*-definable in NU°, where all α_0, $\alpha_{i_1}, \ldots, \alpha_{i_u}$ are different, and NU° proves that there is a formalized *eeb*-definable EF sequence ending with this sequent.

Since the initial sequent is trivial, we discuss the case according to the last inference I to the sequent. We treat only nontrivial cases.

(1) I is a $p\Sigma_1^{1,b}$-induction.

Without loss of generality, we assume that I is of the form

$$\frac{\exists\varphi_b C(b, \varphi_b) \to \exists\varphi_{b+1} C(b+1, \varphi_{b+1})}{\exists\varphi_0 C(0, \varphi_0) \to \exists\varphi_n C(n, \varphi_n)}$$

when n is of the form $t(\vec{a})$. By the induction hypothesis, there is an *eeb*-definable formalized EF-sequence ending with

$$\ulcorner \tilde{C}(b, p^{\alpha_b}) \to \tilde{C}(b+1, p^{\varphi_{b+1}}) \urcorner.$$

Then we introduce new propositional variables

$$p_0^1, \ldots, p_{s(a)}^1, p_0^2, \ldots, p_{s(a)}^2, \ldots, p_0^{t(\vec{a})}, \ldots, p_{s(a)}^{t(\vec{a})}.$$

Then we have *eeb*-definable sequence

$$\begin{aligned}&\ulcorner \tilde{C}(0, p^{\alpha_b}) \to \tilde{C}(1, p^1) \urcorner, \ldots,\\ &\ulcorner \tilde{C}(i, p^i) \to \tilde{C}(1+1, p^{i+1}) \urcorner, \ldots,\\ &\ulcorner \tilde{C}(t(\vec{a}) \dot{-} 1, p^{t(\vec{a}) \dot{-} 1}) \to \tilde{C}(t(\vec{a}), p^{t(\vec{a})}) \urcorner.\end{aligned}$$

Joining these implications by $t(a) \dot{-} 1$ cuts gives a formalized EF-sequence ending with

$$\ulcorner \tilde{C}(0, p^{\alpha_b}) \rightarrow \tilde{C}(t(\vec{a}), p^{t(\vec{a})}) \urcorner.$$

All these operations are *eeb*-definable in NU°.

(2) I is second order $\exists$ right.

Without loss of generality, we assume that I is of the form

$$\frac{\exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}) \rightarrow C(\vec{a}, \alpha_0^{s(a)}, \{x\}(x \leq s(a) \wedge D(x)))}{\exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}) \rightarrow \exists\varphi^{s(a)} C(\vec{a}, \alpha_0^{s(a)}, \varphi^{s(a)})}.$$

By the induction hypothesis, there is a formalized EF-sequence ending with

$$\ulcorner \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_i}) \rightarrow \tilde{C}(\vec{a}, p^{\alpha_1}, \tilde{D}) \urcorner.$$

Then we introduce a new propositional variables $p'_0, \ldots, p'_{s(a)}$ and introduce an extension axioms $\ulcorner p'_i \leftrightarrow \tilde{D}(i) \urcorner$ for $i = 0, \ldots, s(a)$ and make an *eeb*-definable EF-sequence ending with

$$\ulcorner \tilde{B}(\vec{a}, \alpha_0^{s(a)}), \alpha^{s(a)}) \urcorner \rightarrow \ulcorner \tilde{C}(\vec{a}, \alpha_0^{s(a)}), p') \urcorner.$$

(3) I is the contraction right.

Without loss of generality, we assume that I is of the form

$$\frac{\exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}) \rightarrow \exists\psi^{s(a)} C(\vec{a}, \alpha_0^{s(a)}, \varphi^{s(a)}), \exists\varphi^{s(a)} C(\vec{a}, \alpha_0^{s(a)}, \varphi^{s(a)})}{\exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}) \rightarrow \exists\varphi^{s(a)} C(\vec{a}, \alpha_0^{s(a)}, \varphi^{s(a)})}.$$

By the induction hypothesis, there exists a formalized *eeb*-definable EF sequence ending with

$$\ulcorner \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_1}) \leftrightarrow \tilde{C}(\vec{a}, p^{\alpha_0}, p') \vee \tilde{C}(\vec{a}, p^{\alpha_0}, p'') \urcorner.$$

Then we introduce new propositional variables $p'''_0, \ldots, p'''_{s(a)}$ and extension axioms

$$p'''_0 \leftrightarrow (p'_0 \wedge \tilde{C}(\vec{a}, p^{\alpha_0}, p')) \vee (p''_0 \wedge \neg\tilde{C}(\vec{a}, p^{\alpha_0}, p')),$$
$$\ldots,$$
$$p'''_{s(a)} \leftrightarrow (p'_{s(a)} \wedge \tilde{C}(\vec{a}, p^{\alpha_0}, p')) \vee (p''_{s(a)} \wedge \neg\tilde{C}(\vec{a}, p^{\alpha_0}, p')),$$

and construct a formalized *eeb*-definable EF sequence ending with

$$\ulcorner \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_1}) \rightarrow \tilde{C}(\vec{a}, p^{\alpha_0}, p''') \urcorner.$$

(4) I is the contraction left.

Without loss of generality, we assume that I is of the form

$$\frac{\exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}), \exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}) \rightarrow \exists\varphi^{s(a)} C(\vec{a}, \alpha_0^{s(a)}, \varphi^{s(a)})}{\exists\psi^{s(a)} B(\vec{a}, \alpha_0^{s(a)}, \psi^{s(a)}) \rightarrow \exists\varphi^{s(a)} C(\vec{a}, \alpha_0^{s(a)}, \varphi^{s(a)})}.$$

By the induction hypothesis, we have a formalized *eeb*-definable EF sequence ending with

$$\ulcorner \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_i}) \wedge \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_j}) \rightarrow \tilde{C}(\vec{a}, p^{\alpha_0}, p') \urcorner.$$

We substitute p^{α_i} for p^{α_j} in the whole EF sequence and use contraction and get a new formalized *eeb*-definable EF sequence ending with

$$\ulcorner \tilde{B}(\vec{a}, p^{\alpha_0}, p^{\alpha_i}) \rightarrow \tilde{C}(\vec{a}, p^{\alpha_0}, p') \urcorner.$$

Theorem 13. *The construction discussed above produces a formalized eeb-definable EF proof of* $\ulcorner\tilde{A}\urcorner$ *from a proof of an elementary bounded formula A in* $p\Sigma_1^{1,b}$*-part of* $V_1^1(BD)$.

A nice feature of the RSUV isomorphism is that the transformation between the first order theory and the second order theory goes automatically. For the case of TNC° and $NU°$, very small number, small number, sharply bounded, *esb* etc., go to small number, ordinary number, bounded, elementary bounded, *eeb* etc. Besides everything goes automatically.

Take for example Theorem 6 and Theorem 7, there Atom eval, Prf_F, Prf_{EF} eval, S are all *esb* in TNC°. Then the corresponding notions $\tilde{A}$tom eval, $\tilde{P}rf_F$, $\tilde{P}rf_{EF}$, Eval, $\tilde{S}$ are all *eeb* in $NU°$ and we get the following theorems

Theorem 14. $NU°$ *proves that F is a sound proof system i.e.:*

$$\tilde{A}tom\ eval(\alpha^{s(a)}), \tilde{P}rf_F(\beta^{s(a)}, \ulcorner A\urcorner) \to Eval(\tilde{S}(\ulcorner A\urcorner, \alpha^{s(a)})).$$

Theorem 15. $p\Sigma_1^{1,b}$*-part of* $V_1^1(BD)$ *proves that EF is a sound proof system i.e.:*

$$\tilde{A}tom\ eval(\alpha^{s(a)}), \tilde{P}rf_{EF}(\beta^{s(a)}, \ulcorner A\urcorner) \to Eval(\tilde{S}(\ulcorner A\urcorner, \alpha^{s(a)})).$$

Now we have the following theorem.

Theorem 16. *Let A be elementary bounded in* $V_1^1(BD)$. *Then A is provable in* $p\Sigma_1^{1,b}$*-part of* $V_1^1(BD)$ *iff* $\langle A\rangle$ *i.e.* $\ulcorner\tilde{A}\urcorner$ *is provable in uniform EF, where the uniformity is in the sense of* $NU°$.

Proof. If A is provable in $p\Sigma_1^{1,b}$-part of $V_1^1(BD)$, then by Theorem 13 $\ulcorner\tilde{A}\urcorner$ is provable in uniform EF. If $\ulcorner\tilde{A}\urcorner$ is provable in uniform EF, then by Lemma 12, A is provable in $p\Sigma_1^{1,b}$-part of $V_1^1(BD)$. □

By the RSUV isomorphism, we get the following theorem.

Theorem 17. *Let A be sharply bounded in* S_2^1. *Then A is provable in* S_2^1 *iff* $\langle A\rangle$ *i.e.* $\ulcorner\tilde{A}\urcorner$ *is provable in uniform EF.*

Proof. The theorem is obtained from Theorem 16 by the RSUV isomorphism together with the well known fact that A is provable in S_2^1 iff A is provable in uniform $p\Sigma_1^b$-part of S_2^1. □

In [9], P. Colte introduced a free variable equational logic ALV, whose terms represent exactly the NC^1 computable functions, and with the property that if "$f = g$" is provable in ALV then the infinite family $\{|f = g|\,|_m^n : n, m \in \mathrm{N}\}$ of propositional tautologies admits polynomial length Frege proofs. Relating with this, we are going to show that $T°NC°$ and uniform F are isomorphic with respect to NC^1 formulas. Notice that uniform F proof is a polynomial size F proof.

As is discussed in §3, and NC^1 formula is expressed by a sharply bounded formula in $T°NC°$. A sharply bounded formula in $T°NC°$ corresponds to an elementary bounded formula in $N^\circ U^\circ$. For an elementary bounded formula A in $N^\circ U^\circ$ we have already defined $\langle A\rangle$ i.e. $\ulcorner\tilde{A}\urcorner$. Therefore what we have to do for a complete definition of $\langle A\rangle$ for an elementary bounded formula in $N^\circ U^\circ$ is an interpretation of Orand in NU°.

Let A be Orand$(\{x\}D(x), i)$, where i is small. First we make a complete normal $(\vee, \wedge)$-formula (t, f) with height $2(i+1)$ satisfying the following condition.

For the leaf $2^{2(i+1)}+j$ with $j < 2^{2(i+1)}$, $f(j) = p_j$. I.e. $\forall j < 2^{2(i+1)}(f(2^{2(i+1)}+j) = p_j)$. (t, f) is uniquely defined and *eeb*-definable i.e. t and f are expressed by an *eeb*-abstract in NU°. We denote (t, f) by $\ulcorner T(p_0, \ldots, p_{n-1})\urcorner$ where $n = 2^{2(i+1)}$. Let $\langle D(a)\rangle$ be $\ulcorner\tilde{D}(a)\urcorner$. Then $\langle A\rangle$ is defined to be

$$\ulcorner T(\tilde{D}(0), \ldots, \tilde{D}(n-1))\urcorner$$

i.e. $\tilde{A}$ is $T(\tilde{D}(0), \ldots, \tilde{D}(n-1))$ for $\langle A\rangle = \ulcorner\tilde{A}\urcorner$. Obviously $\langle A\rangle$ is *eeb*-definable in NU° and the following lemmas are proved in the same way as before.

Lemma 18. *Let A be elementary bounded in $N^\circ U^\circ$. Then NU° proves the following formula.*

$$\mathrm{Eval}(\tilde{S}(\ulcorner A\urcorner, \{x\}B_0(x))) \leftrightarrow A.$$

Especially if A is an axiom on Orand, then NU° proves $\mathrm{Eval}(\tilde{S}(\ulcorner A\urcorner, \{x\}B_0(x)))$.

Now we simulate a proof in $T^\circ NC^\circ$ in uniform F.

More precisely we consider a proof of an NC^1-formula A i.e. a sharply bounded formula in $T^\circ NC^\circ$. Then there is a proof of A in $T^\circ NC^\circ$ in which every formula is a sharply bounded formula. Therefore we consider only such a proof. Since $T^\circ NC^\circ$ and $N^\circ U^\circ$ are isomorphic by the RSUV isomorphism, we will simulate a proof in $N^\circ U^\circ$ by a uniform EF proof. Let P be a proof in $N^\circ U^\circ$ in which every formula is elementary bounded. As before, we assume that the end sequent of P has only free variable a and all the second order variables in P are $\alpha_0^{s(a)}, \ldots, \alpha_{k-1}^{s(a)}$. Let $A_1, \ldots, A_m \to B_1, \ldots, B_n$ be a sequent in P. Then we prove by induction on the number of inferences to $A_1, \ldots, A_m \to B_1, \ldots, B_n$, there exists a uniform F proof to $\ulcorner\tilde{A}_1 \wedge \ldots \wedge \tilde{A}_m \to \tilde{B}_1 \vee \ldots \vee \tilde{B}_n\urcorner$ where $\ulcorner\tilde{A}_i\urcorner$ and $\ulcorner\tilde{B}_j\urcorner$ are $\langle A_i\rangle$ and $\langle B_j\rangle$ respectively. Since the initial sequent is taken care of by Lemma 18, we discuss the case according to the last inference I to the sequent. We treat only nontrivial cases.

(1) I is $\Delta_0^{1,b}(BD)$-IND.

Without loss of generality we assume that I is of the form

$$\frac{A(a) \to A(a+1)}{A(0) \to A(t)}$$

where $A(a)$ is an elementary bounded formula.

Then by the induction hypothesis there is a formalized F-proof ending with

$$\ulcorner\tilde{A}(a) \to \tilde{A}(a+1)\urcorner.$$

Then we have *eeb*-definable sequence

$$\begin{aligned}&\ulcorner \tilde{A}(0) \to \tilde{A}(1)\urcorner, \dots,\\ &\ulcorner \tilde{A}(i) \to \tilde{A}(i+1)\urcorner, \dots,\\ &\ulcorner \tilde{A}(t \dot{-} 1) \to \tilde{A}(t)\urcorner.\end{aligned}$$

Joining these implications by $t \dot{-} 1$ cuts gives a formalized F-proof ending with

$$\ulcorner \tilde{A}(0) \to \tilde{A}(t)\urcorner.$$

All these operations are *eeb*-definable in $NU°$.

(2) I is $\forall \leq$ right.

Without loss of generality we assume that I is of the form

$$\frac{a \leq t, A \to B(a)}{A \to \forall x \leq t B(x)}.$$

By the induction hypothesis, there is a formalized F-proof ending with

$$\ulcorner a \leq t \wedge \tilde{A} \to \tilde{B}(a)\urcorner.$$

Therefore for $i \leq t$, there is a formalized F-proof ending with

$$\ulcorner \tilde{A} \to \tilde{B}(i)\urcorner.$$

As special cases, we have formalized F-proofs ending with

$$\ulcorner \tilde{A} \to \tilde{B}(0)\urcorner,\ \ulcorner \tilde{A} \to \tilde{B}(1)\urcorner, \dots, \ulcorner \tilde{A} \to \tilde{B}(t)\urcorner.$$

Using $\wedge$ inferences on these, we get a formalized F-proof ending with

$$\ulcorner \tilde{A} \to \bigwedge_{t \leq t} \tilde{B}(i)\urcorner.$$

All these operations are *eeb*-definable in $NU°$.

(3) I is $\forall \leq$ left.

Without loss of generality we assume that I is of the form

$$\frac{A(t) \to B}{t \leq s, \forall x \leq s A(x) \to B}.$$

By the induction hypothesis, there is a formalized F-proof ending with

$$\ulcorner \tilde{A}(t) \to \tilde{B}\urcorner.$$

¿From this we can get a formalized F-proof ending with $\ulcorner 1 \wedge \bigwedge_{i \leq s} \tilde{A}(i) \to \tilde{B}\urcorner$ if $t \leq s$ and $\ulcorner 0 \wedge \bigwedge_{i \leq s} \tilde{A}(i) \to \tilde{B}\urcorner$ if $s < t$. All these operations are *eeb*-definable in $NU°$.

Theorem 19. *The construction discussed above produces a formalized eeb-definable F-proof of $\ulcorner \tilde{A} \urcorner$ from a proof of an elementary bounded formula A in $N^\circ U^\circ$.*

Now we have the following theorem.

Theorem 20. *Let A be elementary bounded in $N^\circ U^\circ$. Then A is provable in $N^\circ U^\circ$ iff $\langle A \rangle$ i.e. $\ulcorner \tilde{A} \urcorner$ is provable in uniform F, where the uniformity is in the sense of NU°.*

Proof. If A is provable in $N^\circ U^\circ$, then there is an $N^\circ U^\circ$-proof of A in which every formula is elementary bounded. Then by Theorem 19, $\ulcorner \tilde{A} \urcorner$ is provable in uniform F. On the other hand, if $\ulcorner \tilde{A} \urcorner$ is provable in uniform F, then by Lemma 18 A is provable in NU° and therefore provable in $N^\circ U^\circ$. □

By the RSUV isomorphism, we get the following theorem.

Theorem 20. *Let A be sharply bounded in $T^\circ NU^\circ$. Then A is provable in $T^\circ NU^\circ$ iff $\langle A \rangle$ i.e. $\ulcorner \tilde{A} \urcorner$ is provable in uniform F.*

Proof. The theorem is obtained from Theorem 20 by the RSUV isomorphism together with the well-known fact that if a sharply bounded A is provable in $T^\circ NC^\circ$, then there exists a $T^\circ NC^\circ$ proof of A, in which every formula is sharply bound. □

The theorems Theorem 16 and Theorem 17 are obtained by using Lemma 12. Now Lemma 18 is a generalization of Lemma 12. By the use of Lemma 18 in the place of Lemma 12, we get the following Theorem 22 and Theorem 23 from the proofs of Theorem 16 and Theorem 17, Theorem 22. Let A be elementary bounded in $N^\circ U^\circ$. Then the translation of A in $p\Sigma_1^{1,b}$-part of $V_1^1(BD)$ iff $\langle A \rangle$ i.e. $\ulcorner \tilde{A} \urcorner$ is provable in uniform EF, where the uniformity is in the sense of NU°.

Theorem 23. *Let A be sharply bounded in $T^\circ NU^\circ$. Then translation of A in S_2^1 is provable in S_2^1 $\langle A \rangle$ i.e. $\ulcorner \tilde{A} \urcorner$ is provable in uniform EF.*

As a corollary we have the following theorem.

Theorem 24. *If* TNC° *and S_2^1 are separated by a NC^1 formula A, then uniform F and uniform EF are separated by $\langle A \rangle$. Here by an NC^1 formula we mean an esb formula in* TNC°.

Proof. We may think that A is a sharply bounded formula in $T^\circ NC^\circ$ and of the form $p\Sigma_1^b$ in S_2^1. Then the provability of $\langle A \rangle$ in uniform F is equivalent to the probability of A in TNC° and the provability of $\langle A \rangle$ in uniform EF is equivalent to the probability of A in S_2^1. □

§5. Razborov's clique, Raz-Wigderson's matching and Karchmer-Wigderson's connectivity.

First we formulate Razborov's clique problem in [20]. For this, let r range over $1, 2, \ldots, N$, i over $1, 2, \ldots, n$ and j over $1, 2, \ldots, n-1$. N is the number of vertices in the graph and we are asking whether n-clique exists in the graph.

The given graph is described by edges $A_{r_1r_2}$ where we assume that $r_1 \neq r_2$ and $A_{r_1r_2} = A_{r_2r_1}$. Razborov's positive test is an existence of an n-clique. I.e. there exist vertices $r'_1, \dots, r'_n$ which form a clique. In order to express this we introduce B_{ir} and express the existence of n-clique by the following statement

$$\bigwedge_{i_1 \neq i_2} \bigwedge_{r} \neg(B_{i_1r} \wedge B_{i_2r}) \quad \text{and}$$

$$\bigwedge_{i_1 \neq i_2} \bigwedge_{r_1 \neq r_2} (B_{i_1r_1} \wedge B_{i_2r_2} \supset A_{r_1r_2}).$$

Razboron's negative test is an assignment of a color from $\{1, 2, \dots, n-1\}$ to each vertices such that there are no edge between vertices with the same color. In order to express this we introduce C_{jr} and express the existence of such coloring by the following statement.

$$\bigwedge_{r} \bigvee_{j} C_{jr} \quad \text{and} \quad \bigwedge_{r_1 \neq r_2} \bigwedge_{j} (C_{jr_1} \wedge C_{jr_2} \supset \neg A_{r_1r_2}).$$

Therefore if a positive test works, then any negative test fails.

This is expressed by the following sequent

$$\Gamma \rightarrow \Delta$$

where Γ is the sequent of

$$\{\bigvee_{r} B_{ir}\}_i, \quad \{\bigvee_{j} B_{jr}\}_r$$

$$\{B_{i_1r_1} \wedge B_{i_2r_2} \supset A_{r_1r_2}\}_{i_1 \neq i_2, r_1 \neq r_2}$$

and Δ is the sequence of

$$\{B_{i_1r} \wedge B_{i_2r}\}_{i_1 \neq i_2, r}, \{C_{jr_1} \wedge C_{jr_2} \wedge A_{r_1r_2}\}_{r_1 \neq r_2}.$$

As is discussed in the introduction, S. Buss proved in [4] that the following form of PHP is polynomial size F provable

$$\{\bigvee_{j} P_{ij}\}_i \rightarrow \{P_{i_1j} \wedge P_{i_2j}\}_{j, i_1 \neq i_2}.$$

$\Gamma \rightarrow \Delta$ is polysize F provable since it is reduced to PHP by putting $P_{ij} = \bigvee_r (B_{ir} \wedge C_{jr})$.

In Raz-Wigderson's example in [19], $N = 3n$ is the number of vertices of a graph. Qestion is whether there exists a matching of m edges. Let r range over $1, 2, \dots, N$, i over $1, 2, \dots, n$ and j over $1, 2, \dots, n-1$. We express a matching of m edges of M_{ir} satisfying

$$\bigwedge_{i} \bigvee_{r_1 \neq r_2} (M_{1r_1} \wedge M_{ir_2}) \quad \text{and}$$

$$\bigwedge_{i_1 \neq i_2} \neg \bigvee_{r} (M_{i_1r} \wedge M_{i_2r}).$$

Now q is a $(n-1)$ vertices in the graph which we express by Q_{jr} satisfying

$$\bigwedge_j \bigvee_r Q_{jr}, \quad \bigwedge_j \bigwedge_{r_1 \neq r_2} \neg(Q_{jr_1} \wedge Q_{jr_2})$$

$$\text{and} \quad \bigwedge_{j_1 \neq j_2} \neg \bigvee_r (Q_{j_1 r} \wedge Q_{j_2 r}).$$

Then we have a conclusion that there exists $r_1 \neq r_2$ such that $\bigvee_i(M_{ir_1} \wedge M_{ir_2}) \wedge \bigwedge_i(\neg Q_{jr_1} \wedge \neg Q_{jr_2})$. I.e. the following sequent is a tautology.

$\Gamma, \Delta, A \rightarrow$

where Γ is $\{\bigvee_{r_1 \neq r_2}(M_{ir_1} \wedge M_{ir_2})\}_i, \{\neg \bigvee_r(M_{i_1 r} \wedge M_{i_2 r})\}_{r_1 \neq r_2}$,

Δ is $\{\bigvee_r Q_{jr}\}_j, \{\neg(Q_{jr_1} \wedge Q_{jr_2})\}_{r_1 \neq r_2}$,

$\{\bigvee_r Q_{j_1 r} \wedge Q_{j_2 r})\}_{j_1 \neq j_2}$

and A is $\bigwedge \neg(M_{ir_1} \wedge M_{ir_2}) \vee \bigvee_j(Q_{jr_1} \vee Q_{jr_2})\}_{r_1 \neq r_2}$.

This sequent is polysized F provable as a special case of PHP by putting P_{ij} to be

$$\bigvee_r (M_{ir} \wedge Q_{jr}).$$

In [14] and [15], Krchmer and Wigderson proved that st connectivity problem is not monotone NC^1. The implicit definition of their connectivity problem separates not only constant depth polynomial size F proof and cut-free polynomial size LK but also cut-free polynomial size LK with substitution and cut-free polynomial size LK.

Let G be a graph and $0, 1, \ldots, n$ be vertices of G. We use the following propositional variables.

$A_{ij} = A_{ji}$ for $0 \leq i, j \leq n$ and $i \neq j$.

B_{ij} for $0 \leq i, j \leq n$.

C_{sj} for $s = 1, 2$ and $0 \leq i \leq n$.

The intended meaning of A_{ij} is "(i,j) is an edge in G." We define two formulas Φ and Ψ as follows.

Φ is $\bigwedge_i \bigvee_j B_{ij} \wedge \bigwedge_{r<n} \bigwedge_{i \neq j}(B_{ri} \wedge B_{r+1j} \rightarrow A_{ij}) \wedge B_{00} \wedge B_{nn}$.

Ψ is $\bigwedge_j(C_{1j} \wedge C_{2j}) \wedge \bigwedge_{i,j}(C_{1j} \wedge C_{2j} \rightarrow \neg A_{ij}) \wedge C_{10} \wedge C_{2n}$.

The meaning of Φ is "0 and n are connected in the graph" and the meaning of Ψ is "$\{0,1,2,\ldots,n\}$ is a union of $G_1 = \{i \mid C_{1i}\}$ and $G_2 = \{i \mid G_{2i}\}$, $0 \in G_1$, $n \in G_2$, and for every $i \in G_1$ and $j \in G_2$, there are no edge (i,j) in G." Therefore $\exists \vec{B} \Phi(\vec{B}, \vec{A}) \rightarrow \neg \exists \vec{C} \Psi(\vec{C}, \vec{A})$. Then $\Phi \rightarrow \neg \Psi$ is provable and $\Phi \rightarrow \neg \Psi$ is our candidate to separate polynomial size F and polynomial size EF. Here we are going to prove that $\Phi \rightarrow \neg \Psi$ has a polynomial size EF proof. However we use SF in the place of EF.

Both SF and EF are introduced in Cook-Reckhow [12].

The substitution rule allows to substitute in one inference step simultaneously formulas for atoms:

$$\frac{\mathcal{O}(p_1, \ldots, p_n)}{\mathcal{O}(\varphi_1, \ldots, \varphi_n)}.$$

A Frege system argumented by the substitution rule is denoted by SF.

Reckhow proved in [21] that any Frege system polynomially simulate any other Frege system. Cook, Dowd, and Krajíček-Pudlák proved that EF and SF polynomially simulate each other. See Krajíček [17] for a comprehensive treatment on the subject.

Now we assume $n = 2^d$. We define $\Phi(s, i_0, j_0)$ for $s \leq d$ to be

$$\bigwedge_{1\leq 2^s} \bigvee_{j} B_{ij} \wedge \bigwedge_{r<2^s} \bigwedge_{i\neq j} (B_{ri} \wedge B_{r+1j} \rightarrow A_{ij}) \wedge B_{0i_0} \wedge B_{2^s j_0}.$$

We define $\Psi(i_0, j_0)$ to be

$$\bigwedge_{j} (C_{1j} \vee C_{2j}) \wedge \bigwedge_{i,j} (C_{1j} \wedge C_{2j} \rightarrow \neg A_{ij}) \wedge C_{1i_0} \wedge C_{2j_0}.$$

The sizes of $\Phi(s, i_0, j_0)$ and $\Psi(i_0, j_0)$ are polynomial of n.

We prove by induction on s that there is a polynomial size SF proof of

$$\Phi(s, i_0, j_0) \rightarrow \neg\Psi(i_0, j_0).$$

If $s = 0$, then B_{0i_0} and B_{ij0}. Therefore $\Phi(s, i_0, j_0)$ implies $A_{i_0j_0}$ which contradicts $\Psi(i_0, j_0)$.

Now suppose that $\Phi(s, i_0, j_0) \rightarrow \neg\Psi(i_0, j_0)$ has a polynomial size SF proof for s, i_0, j_0. We are going to construct a polynomial size SF proof of $\Phi(s + 1, i_0, j_0) \rightarrow \neg\Psi(i_0, j_0)$. From $\Phi(s + 1, i_0, j_0)$, there exists r such that $B_{2^s r}$. Assuming $\Psi(i_0, j_0)$, either C_{1r} or C_{2r} holds.

Case 1. C_{2r} holds.

In this case $\Phi(s + 1, i_0, j_0) \wedge B_{2^s r}$ implies $\Phi(s, i_0, r)$. $\Psi(i_0, j_0) \wedge C_{2r}$ implies $\Psi(i_0, r)$. By the induction hypothesis $\Phi(s, i_0, r) \rightarrow \neg\Psi(i_0, r)$ has a polynomial size SF proof.

Case 2. C_{1r} holds.

We make a simultaneous substitution of $B_{2^s+i\ j}$ for B_{ij} for all $i \leq 2^s$, $j \leq n$ in

$$\Phi(s, r, j_0) \rightarrow \neg\Psi(r, j_0).$$

Then we have $\tilde{\Phi}(s, r, j_0) \rightarrow \neg\Psi(r, j_0)$. It is easily seen that $\Phi(s+1, i_0, j_0) \wedge B_{2^s r}$ implies $\tilde{\Phi}(s, r, j_0)$ and $\Psi(i_0, j_0) \wedge C_{1r}$ implies $\Psi(r, j_0)$. Therefore we have a polynomial size SF proof of

$$\Phi(s + 1, i_0, j_0) \wedge C_{1r} \rightarrow \neg\Psi(i_0, j_0).$$

Let $f(s)$ be the size of the proof of $\Phi(s, i_0, j_0)$. Then the proof stated above implies

$$f(s + 1) \leq f(s) + p(n)$$

where $p(n)$ is a fixed polynomial of n. Therefore we get $f(x) \leq sp(n) \leq np(n)$. Therefore we have the following theorem.

Theorem 25. *$\Phi \rightarrow \neg\Psi$ has a polynomial size EF proof.*

Now we are going to introduce a special form of Gentzen's LK and prove that a formulation of $\Phi \rightarrow \neg\Psi$ in LK has a polynomial size cut-free proof in LK with substitution which is denoted by SLK but it does not have a polynomial size cut-free LK proof.

We make a following slight modification of Gentzen's LK. In original LK, the principal formula and the auxiliary formula of the inference are always at the end of sequent e.g.

$$\frac{\Gamma \rightarrow \Delta, A}{\neg A, \Gamma \rightarrow \Delta}.$$

We generalize every inference-schema so that the principal formula and the auxiliary formula may not be at the end of the sequent. E.g. the inference $\neg$ left is now of the form

$$\frac{\Gamma, \Pi \rightarrow \Delta, A, \Lambda}{\Gamma, \neg A, \Pi \rightarrow \Delta, \Lambda}.$$

Therefore we can eliminate the inference exchange from LK. We call this modified LK simply by LK.

Now we rewrite $\Phi \rightarrow \neg\Psi$ in the following form

$$\Gamma_1, \Gamma_2 \rightarrow$$

where Γ_1 is

$$\{\bigvee_j B_{ij}\}_i, \{B_{r,i} \wedge B_{r+1,j} \supset \neg A_{ij})\}_{\substack{r<n \\ i \neq j}}, B_{1,1}, B_{n,n}$$

and Γ_2 is

$$\{C_{1,j} \vee C_{2,j}\}_j, \{C_{1,j} \wedge C_{2,j} \supset \neg A_{ij}), C_{1,1}, C_{2,n}.$$

It is easily seen from the previous proof that Γ_1, $\Gamma_2 \rightarrow$ has a polynomial size cut-free SLK proof, where the substitution rule can be written as

$$\frac{\Gamma(p_1, \ldots, p_n) \rightarrow \Delta(p_1, \ldots, p_n)}{\Gamma(\phi_1, \ldots, \phi_n) \rightarrow \Delta(\phi_1, \ldots, \phi_n)}.$$

Now we also show that $\Gamma_1, \Gamma_2 \rightarrow$ has constant depth polynomial size F proof. We always assume Γ_1 and Γ_2 and prove our statement in 4 steps.

1. $\displaystyle\bigvee_{i<i} \left(\bigwedge_{j<i} B_{j1} \wedge \bigvee_{k \neq 1} B_{ik} \right)$

Assume its negation $\displaystyle\bigwedge_{1<i} \left(\bigwedge_{j<i} B_{j1} \supset \bigwedge_{k \neq 1} \neg B_{ik} \right)$. Then we get $\wedge_{k\neq 1} \neg B_{ik}$ by induction on i, whence we get B_{i1}. As its special case we get $B_{n-1\,1}$. This together with B_{nn} implies A_{1n}. On the other hand we get $\neg A_{1n}$ from C_{11} and C_{2n} which is a contradiction.

2. $\neg C_{21}$.

By 1, there exists $i > 1$ such that $\bigwedge_{j<i} B_{j1} \wedge \bigvee_{k \neq 1} B_{ik}$. Now there exists $k \neq 1$ such that B_{ik}. Now suppose C_{21}. If C_{1k} holds, then $\neg A_{1k}$. On the other hand, $B_{i-1\ 1} \wedge B_{ik}$ implies A_{1k} whence follows a contradiction. Now assume that C_{2k} holds. Then C_{11} and C_{2k} implies $\neg A_{1k}$ whence follows a contradiction again.

3. $\bigwedge_{i} \bigvee_{k} (B_{ik}) \wedge \neg C_{2k})$.

We prove $\bigvee_{k} (B_{ik}) \wedge \neg C_{2k})$ by induction on i. If $i = 1$, then $B_{11} \wedge \neg C_{21}$ because of 2. Now we assume $B_{ik} \wedge \neg C_{2k}$. Then we have C_{1k}. Suppose that $B_{i+1k'} \wedge C_{2k'}$. Then $k \neq k'$ since $\neg C_{2k}$ and $C_{2k'}$. Then $A_{kk'}$ since B_{ik} and $B_{i+1k'}$ and $\neg A_{kk'}$ since C_{1k} and $C_{2k'}$ which is a contradiction.

4. Now we show a contradiction.

Take i to be $n-1$ in $\bigvee_{k} B_{ik} \wedge \neg C_{2k}$ in 3. Then we have $\bigvee_{k} B_{n-1k} \wedge \neg C_{2k}$. Let k satisfy $B_{n-1}k \wedge \neg C_{2k}$. Then we have $k \neq n$ since $\neg C_{2k}$ and C_{2n}. Then we have A_{kn} since B_{n-1k} and B_{nn}. On the other hand, we have $\neg A_{kn}$ since C_{1k} and C_{2n}. This is a contradiction.

It is easily seen that this proof can be converted to constant depth polynomial size F proof.

Now we are going to show that $\Gamma_1, \Gamma_2 \rightarrow$ does not have any polynomial size cut-free LK proof. We use communication complexity in [14].

Let $f : \{A_{ij} \mid 1 \leq i, j \leq n \text{ and } i \neq j\} \rightarrow \{0,1\}$ be a Boolean formula expressing that the vertices 1 and n are connected in the graph. Then the result in [14] and [15] implies that the monotone formula size of f $L_m(f)$ is superpolynomial. Let $B_1 \in f^{-1}(1)$ and $B_0 \in f^{-1}(0)$. Let D be a deterministic protocal such that for any given B_1, B_0 as above, two players I and II find A_{ij} by following the protocal D such that $A_{ij} \in B_1$ and $A_{ij} \notin B_0$. Then Theorem 3.3.2 in [14] implies that the number of histories of D is greater than $L_m(f)$ therefore superpolynomial in n.

Now let P be a cut-free LK proof of

$$\Gamma_1, \Gamma_2 \rightarrow .$$

We are going to define a protocal D from P. Let B_1 and B_0 satisfy $B_1 \in f^{-1}(1)$ and $B_0 \in f^{-1}(0)$, and G_1 and G_2 be the graphs expressed by B_1 and B_0 respectively. Then there exists a path from 1 to n in G_1. By repeating some vertices if necessary, we can find a path $i = i_1 - i_2 - i_3 - \cdots - i_n = n$ in G. Then we assign true to B_{ri} iff $i = i_r$. On the other hand, we can find C_1 and C_2 for G_2 such that $C_1 \cup C_2 = \{1, 2, \ldots, n\}$ and for any $i \in C_1$ and $j \in C_2$, G_2 has no edge (ij) and $1 \in C_1$ and $n \in C_2$. Then we assign true to C_{rk} iff $r = 1 \wedge k \in C_1$ or $r = 2 \wedge k \in C_2$. The player I has the perfect information on G_1 therefore on assignment on B_{r_i} and the player II has the perfect information on G_2 therefore on assignment on C_{rk}.

Now we make the protocal D as follows. Look the proof P from the bottom and let the logical inference J be the bottom most logical inference to the sequent considered in P. We define the protocal according to J

Case (1). J is of the form

$$\frac{\bigvee_{j1} B_{ij1}, \Gamma \to \Delta \quad \bigvee_{j2} B_{ij2}, \Gamma \to \Delta}{\bigvee_{j} B_{ij1}, \Gamma \to \Delta}$$

where $\bigvee_{j} B_{ij1}$ is $(\bigvee_{j1} B_{ij1}) \vee (\bigvee_{j2} B_{ij2})$. In this case I checks which of $\bigvee_{j1} B_{ij1}$, and $\bigvee_{j2} B_{ij2}$ is true. If $\bigvee_{j1} B_{ij1}$, is true, then I sends a message that he picks the sequent $\bigvee_{j1} B_{ij1}, \Gamma \to \Delta$ if $\bigvee_{j1} B_{ij1}$ is true or the sequent $\bigvee_{j2} B_{ij2}, \Gamma \to \Delta$ if $\bigvee_{j2} B_{ij2}$ is true.

Case (2). J is of the form

$$\frac{A_{ij}, \Gamma \to \Delta \quad \Gamma \to \Delta, B_{r,i} \wedge B_{r+1,j}}{B_{r,i} \wedge B_{r+1,j} \supset A_{ij}, \Gamma \to \Delta}.$$

In this case, I checks whether $B_{r,i} \wedge B_{r+1,j}$ is true or not. if $B_{r,i} \wedge B_{r+1,j}$ is true, then I sends a message that he chooses the sequent A_{ij}, $\Gamma \to \Delta$. Otherwise I sends a message that he chooses the sequent

$$\Gamma \to \Delta, B_{r,i} \wedge B_{r+1,j}$$

Case (3). J is of the form

$$\frac{\Gamma \to \Delta, B_{r,i}, \quad \Gamma \to \Delta, B_{r+1,j}}{\Gamma \to \Delta, B_{r,i} \wedge B_{r+1,j}}.$$

In this case $B_{r,i} \wedge B_{r+1,j}$ must be false. If $B_{r,i}$ is false, then I sends a message that he chooses $\Gamma \to \Delta$, $B_{r,i}$. Otherwise $B_{r+1,j}$ is false and I sends a message that he chooses $\Gamma \to \Delta$, $B_{r+1,j}$.

Case (4). J is of the form

$$\frac{C_{1,j}, \Gamma \to \Delta \quad C_{2,j}, \Gamma \to \Delta}{C_{1,j} \vee C_{2,j}, \Gamma \to \Delta}.$$

In this case, II checks whether $C_{1,j}$ is true or not. If $C_{1,j}$ is true, *II* sends a message that he chooses $C_{1,j}$, $\Gamma \to \Delta$. Otherwise $C_{2,j}$ is true and *II* sends a message that he chooses $C_{2,j}$, $\Gamma \to \Delta$.

Case (5). J is of the form

$$\frac{\neg A_{ij}, \Gamma \to \Delta \quad \Gamma \to \Delta, C_{1,j} \wedge C_{2,j}}{C_{1,j} \wedge C_{2,j} \supset \neg A_{ij}, \Gamma \to \Delta}.$$

In this case, II checks whether $C_{1,j} \wedge C_{2,j}$ is true or not. If $C_{1,j} \wedge C_{2,j}$ is true, II sends a message that he chooses the sequent $\neg A_{ij}$, $\Gamma \to \Delta$. Otherwise II sends a message that he chooses the sequent

$$\Gamma \to \Delta, C_{1,j} \wedge C_{2,j}.$$

Case (6). J is of the form

$$\frac{\Gamma \to \Delta, C_{1,j} \quad \Gamma \to \Delta, C_{2,j}}{\Gamma \to \Delta, C_{1,j} \wedge C_{2,j}}\,.$$

In this case $C_{1,j} \wedge C_{2,j}$ must be false. If $C_{1,j}$ is false, then II sends a message that he chooses the sequent $\Gamma \to \Delta,\ C_{1,j}$. Otherwise $C_{2,j}$ must be false and II sends a message that he chooses the sequent $\Gamma \to \Delta,\ C_{2,j}$.

Case (7). J is of the form

$$\frac{\Gamma \to \Delta, A_{ij}}{\neg A_{ij}, \Gamma \to \Delta}\,.$$

In this case I and II simply go up to $\Gamma \to \Delta,\ A_{ij}$.

Suppose $\Gamma \to \Delta$ comes up when we use the protocal D. If B_{ij} is in Γ, then B_{ij} must be true and if B_{ij} is in Δ, then B_{ij} must be false. In the same way if C_{ij} is in Γ, then C_{ij} must be true and if C_{ij} is in Δ, then C_{ij} must be false.

We assume that all the initial sequents are atomic. Then when D is terminated, we must end up a sequent of the form

$$A_{ij} \to A_{ij}.$$

It is easily seen that A_{ij} is the graph G_1 but not in the graph G_2.

The number of all histories of $D \leq$ the number of all branches in $P \leq$ the number of initial sequents in P.

Therefore the size of P must be greater than $L_m(f)$ and therefore superpolynomial.

References

1. D. A. Mix Barrington, N. Immerman and H. Straubing, *On uniform with NC^1, Structure in Complexity Theory*, Third Annual Conf., IEEE Computer Society Press (1988) pp. 47–59, to appear in J. Comp. Syst. Sci..
2. S. R. Buss, *Bounded Arithmetic*, Napoli, Bibliopolis 1986.
3. ———, *The Boolean formula value problem is in ALOGTIME*, In Proceedings of the 19-th Annual ACM Symposium on Theory of Computing, pp. 123–131, 1987.
4. ———, *Polynomial size proofs of the propositional pigeonhole principle*, J. Symbolic Logic **52** (1987), 66–92.
5. ———, *Algorithms for Boolean formula evaluation and for tree-contraction*, In Proof Theory, Complexity and Arithmetic (P. Clote and J. Krajíček, eds.), Oxford University Press, pp. 95–115.
6. ———, *Propositional consistency proofs*, Annals of Pure and Applied Logic **52** (1991), 3–29.
7. S. R. Buss, S. A. Cook, A. Gupta and V. Ramachandran, *An optimal parallel algorithm for formula evaluation*, SIAM J. on Computing **21** (1992), 755–780.
8. P. Clote, *Sequential, machine-independent characterizations of the parallel complexity classes ALOGTIME, AC^k, NC^k and NC,*, Feasible Mathematics (S. R. Buss and P. Scott, eds.), Birkhäuser, 1990, pp. 49–70.
9. P. Clote, *ALOGTIME and a conjecture of S. A. Cook*, Annals of Mathematics and Artificial Intelligence **6** (1992), 57–106, extended abstract in proceedings of IEEE Logic in Computer Science, Philadelphia, June 1990..

10. P. Clote and G. Takeuti, *First order bounded arithmetic and small boolean circuit compleity class*, in Feasible Mathematics II (P. Clote and J. Remmel, eds.), Birkhäuser, 1995, pp. 154–218.
11. S. A. Cook, *Feasibly constructive proofs and the propositional calculus*, Proc. 7-th ACM Symp. on the Theory of Computation (1975), 83–97.
12. S. A. Cook and R. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic **44** (1977), 36–50.
13. M. Dowd, *Propositional representation of arithmetical proofs*, Ph.D. Dissertation, University of Toronto, Department of Computer Science Technical Report 132/79, April 1979.
14. M. Karchmer, *Communication Complexity: A New Approach to Circuit Depth*, The MIT Press, 1989.
15. M. Karchmer and A. Wigderson, *Monotone circuits for connectiveity require sur- logarithmic depth*, Proc. 20-th Annual ACM symp. on Theory of Computation, ACM Press, 1988, pp. 539–550.
16. J. Krajíček, *On Frege and Extended Frege Proof Systems*, Feasible Mathematics II (P. Clote and J. Remmel, eds.), Birkhäser, 1995, pp. 284–319.
17. ———, *Bounded Arithmetic, Propositional Logic and Complexity Theory*, Cambridge University Press (to appear).
18. J. Krajíček and P. Publák, *Propositional proof systems, the consistency of first order theories and the complexity of computations*, J. Symbolic Logic **54** (1989), 1063–1079.
19. R. Raz and A. Wigderson, *Monotone circuits for matching require linear depth*, Proc. 22nd Ann. ACM Symp. on Theory of Computing (1990), 287–292.
20. A. A. Razborov, *Lower bounds on the monotone complexity of some Boolean functions*, Dokl. Akad. Nauk SSSR **281(4)** (1985), 798–801; English translation in: Soviet Math. Dokl. **31** (1985), 345–357.
21. R. A. Reckhow, *On the lengths of proofs in the proposition calculus*, Ph.D. Thesis 1976, Department of Computer Science, University of Toronto, Technical Report 87.
22. G. Takeuti, S^i_3 *and* $\overset{o}{V}{}^i_2(BD)$, Archive for Mathematical Logic **29** (1990), 149–169.
23. ———, *RSUV Isomorphisms*, Arithmetic, Proof Theory and Computational Complexity (P. Clote and J. Krajíček, eds.), Oxford Univ. Press, 1993, pp. 364–386.
24. ———, *RSUV Isomorphisms for* TAC^i, TNC^i *and TLS* (to appearArchive for Mathematical Logic).

Department of Mathematics, University of Illinois, Urbana, Illinois 61801, U.S.A.

Characterizing Parallel Time by Type 2 Recursions with Polynomial Output Length

Stephen J. Bellantoni

Abstract. Recent work in recursion theory has shown that a significant number of traditional complexity classes can be defined using a ramified form of recursion, in which recursion is not allowed on values that themselves were defined by a recursion. Applications of this technique to parallel complexity classes have resulted in function classes that have outputs that are not length-bounded by any fixed polynomial. The current work shows that appropriate tiering of the initial functions reduces the output size to a polynomial; at the same time, type 2 recursion allows one to access the results of preceding computations without passing through an encoding.

Traditional recursion theory can be ramified by typing the inputs and outputs of functions into "tiers" numbered 0, 1, 2, etc: each recursively-defined function produces an output value one tier lower than the recursion variable. Remarkably, this structuring of recursion gives characterizations of computational complexity classes originally defined in terms of explicit resource bounds. See Simmons [17], Bellantoni and Cook [1], Leivant [9], [10], [11], Bellantoni [3], [2], Leivant and Marion [12], [13], and Bloch [4] for characterizations.

In the area of parallel computational complexity, Bloch [4] characterized the functions computable by an alternating Turing machine in $O(\log^k n)$ time with no space bound ("parallel polylog time"). The characterization uses ordinary type level 1 recursion together with a pairing function to code together the results of computations. An essentially similar characterization was implicit in Leivant's subsequent result [10]. One drawback of these results is the fact that the output of the functions is not polynomially bounded. Indeed this is intrinsic to the technique used, which requires coding together the results of subcomputations in such a way as to construct a computation tree of depth $\Omega(\log^k n)$.

A type 1 characterization of the parallel class NC is given in [2], based on Clote's work [6]. A defect of this characterization is that it refers explictly to polynomials by including the # initial function.

Immerman's well-known results [8] giving first-order characterizations indicates that there is a connection between type 2 constructs and parallel computation. These non-ramified characterizations use an explicit bound on the closure ordinal of the fixed-point operation. Since this work characterizes relations rather than functions, it is not intended to address the question of function growth rate.

Ramified second-order constructs were used by Leivant [11] and Leivant and Marion [12], [13] to characterize the PSPACE computable functions and the elementary-time computable functions. There is also a characterization of the parallel polylog time computable functions implicit in [13]. Although coding is not used to prove this characterization, the function outputs still may be superpolynomial in length. A helpful innovation in this work is the admission as inputs to functionals only those functions that have *predicative function type*: the admissible function's inputs are of higher tier than its output.

The purpose of the present work is to demonstrate techniques for significantly reducing the size of the computed values without changing the complexity of the computation that produces each bit of the value. Specifically, a ramified recursion class is defined such that all the functions are output length-bounded by a polynomial and, regarded bitwise, the defined relations are exactly those computable by an alternating Turing machine in polylog time with no space bound.

The inputs to this class of operations are presented as balanced binary trees. It is shown that if the inputs are instead presented as totally unbalanced trees, then the induced function class consists of the PSPACE computable functions.

A few technical novelties are used to obtain the characterizations. First, "admissible" function types are restricted so that they have inputs of tier at most 1. At the same time, these function inputs can only appear when the functional is defined either by composition or by higher type recursion on a tier 2 variable: higher-type recursion is not allowed on a tier 1 variable. Hence, the recursion variable is always of higher tier than the inputs to any of the parameter functions.

Second, the tree constructors and destructors are defined to have mixed-tier inputs. For example, where earlier works allowed the construction of a tier 0 tree from two other tier 0 trees, we now require that one of the input subtrees have tier 1. These constructors, which are "more predicative" than the earlier ones, do not allow the construction of large output values. The crucial factor in controlling output size is the type of the constructors, rather than the number of critical recursion terms (i.e. rather than the difference between tree recursion and recursion on notation).

To carry out one direction of the proof, functionals E_k^h are defined that encapsulate direct recursion over a tree of height $\Omega(\log^k n)$. This derived recursion scheme has at each base case the evaluation of a function on a $\log^k n$ length path that leads to a leaf of the tree. The base cases are followed by step cases consisting of the application of a step function at each interior node of the tree.

To carry out the other direction of the proof, we ascribe tiers to the gates of a circuit, including oracle gates. Equivalently we can assign tiers to ATM computation states. To the author's knowledge, this is the first time ramified constructs have been applied directly to machine states arising during the course of a computation.

1 Background

Consider nonempty binary trees with leaves labeled with 0 or 1. For a tree u, define the *length* of u by $|u|$ = the number of leaves in the tree. The *node height* of the tree is $\lceil u \rceil$: it is the number of nodes in a maximal root-to-leaf path in the tree. (This differs by 1 from the usual definition of height; note $\lceil u \rceil \geq 1$ for all u.) For integer m, define the *length* of m by $|m| = \max\{1, \lceil \log_2(m+1) \rceil\}$; thus $|m| \geq 1$.

Each tree t *is a presentation of* an integer value $\{t\}$ as follows: (1) if t has length 1, then it denotes the value 0 or 1 according as the leaf's label is 0 or 1; (2) otherwise, t has a left subtree u and a right subtree v and the tree denotes $\{u\} \cdot 2^{|\{v\}|} + \{v\}$.

Each integer value can be presented as any one of infinitely many trees. A *presentation system* P is an assignment of a canonical presentation $P(m)$ for each integer value m, together with a set of *concrete operations* over presentations. The operations are described later. The *normal presentation of* 0 is a single leaf labeled 0, and the *normal presentation of* 1 is a single leaf labeled 1. A binary presentation system is *normal* if it uses the normal presentations for 0 and 1 and satisfies $|P(m)| = |m|$ for all integers m; furthermore, for the system to be normal we require that $m \neq n \Leftrightarrow P(m) \neq P(n)$.

Let the presentation system B ("balanced presentation") be the normal presentation system in which $m > 1$ is presented as a balanced tree whose left child presents $\lfloor m/2^{\lceil |m|/2 \rceil} \rfloor$ and whose right child presents $m \bmod 2^{\lfloor |m|/2 \rfloor}$. For comparison, let the presentation system S "skewed presentation") be the normal presentation system in which $m > 1$ is presented as a totally unbalanced tree whose left child presents $\lfloor m/2 \rfloor$ and whose right child is the normal presentation of $m \bmod 2$.

A class of operations on trees, $2T^+$, is defined below. Let $B/2T^+$ be the set of functions $F : N^k \to N$ such that there is an operation $f \in 2T^+$,

$$F(\overline{n}) = \{f(B(\overline{n}))\}$$

for all $\overline{n}$. The functions $S/2T^+$ are defined analogously.

Note the distinction between "function" and "operation": a function is a mapping on abstract mathematical objects, while an operation is a process performed on concrete presentations of these abstractions. In order to describe an operation one usually requires an equivalence relation on physical presentations; the equivalence classes can be inferred from the denotation operator $\{\cdot\}$.

The use of presentations is not equivalent to simply computing over terms of an algebra, because the presentation system always selects one of the many different possible presentations of the input integer. In contrast, computation over a term algebra does not refer to the natural numbers. Computations over a term algebra cannot be compared to computations of Turing machines without passing through a mapping between the terms and the TM input tapes. Implied in such a mapping is a presentation system for the term algebra and a presentation system for the TM.

The term "computation" refers to the overall system — the computing device plus the presentational context provided by the subject. "Computational complexity" refers, as it always has, to the complexity of the computed function. It is quite different from "operational complexity", the complexity of the mechanical operation performed by the computing device. For example, there are very complex operations that in the end induce a constant function under every presentation. We can also consider "presentational complexity", the complexity of the subject's mapping from objects of attention (abstract integers) to presentations (concrete trees). A mathematical formalization of this concept of presentational complexity would require dealing with a radical difference between the abstract and the concrete, and probably can only be achieved by shunting aside philosophical issues.

2 The operations $2T^+$

The key to the following definition of $2T^+$ is that recursive terms can only appear in "lower tier" positions than the recursion variable, thereby preventing the step function from applying the same tier of recursion to the recursive term. Unlike the earlier systems, $2T^+$ allows complete access to a function defined by the preceding recursion step. This feature requires the use of functions as inputs. I begin with definitions for types and vectors.

The type hierarchy used in the discussion is defined in the usual way, but starting with three ground types T_0, T_1, and T_2 each consisting of the set of nonempty binary trees. Level 0 types consist of trees; level 1 types are operations on trees; and level 2 types are operationals on operations and trees. Product types are identified with function types: for example, type $(\sigma_1 \times \sigma_2) \rightarrow \sigma_3$ is identified with type $\sigma_1 \rightarrow (\sigma_2 \rightarrow \sigma_3)$. One can always assume that the output type is level 0, that is, $\#\overline{\sigma}_{\#\overline{\sigma}} = 1$. We often regard level 0 values (trees) as being operations with arity zero; *strictly level 1* means level 1 with arity greater than zero. Finally, the *level 1 section* of a class is the subset consisting of strictly level 1 functions.

Vector notation is used freely: $\overline{\sigma} \equiv [\sigma_1, \ldots, \sigma_n]$ where $n = \#\overline{\sigma}$. In this paper "#" is used for vector length, not for the smash function. The subscript "o" removes "[" and "]"; thus "$[\overline{\sigma}_\circ, \overline{\tau}_\circ]$" means the concatenation of $\overline{\sigma}$ and $\overline{\tau}$. If $\overline{g}$ is a vector of functions and $\overline{v}$ is a vector of inputs, then $(\overline{gv})$ means $(g_1\overline{v}), \ldots, (g_{\#\overline{g}}\overline{v})$. In some places a semicolon ";" is used in a vector; its meaning is the same as ",".

The *tier* of types is defined by: T_0 is tier 0; T_1 is tier 1; T_2 is tier 2; and the tier of $\overline{\sigma}$ is the tier of the last element, $\sigma_{\#\overline{\sigma}}$.

Inputs are restricted to admissible types:

Definition 1. A type $\overline{\sigma}$ is *admissible* if either $\#\overline{\sigma} = 1$ or else $\overline{\sigma}$ is tier 0 and all σ_i, $1 \leq i < \#\overline{\sigma}$ are tier 1. In other words, admissible inputs are trees or else operations mapping tier 1 trees to tier 0 trees.

The admissible types with nonzero arity are all "predicative function types" in the sense of Leivant and Marion [13], because all the inputs have tier greater than the output. I further require that the function inputs have lower tier than the tier of the variables used in higher-type recursion.

The following "tiered application" conventions will simplify the notation.

(1) Suppose f has type $\overline{\sigma}$ with $\#\overline{\sigma} = k+1$. Usually, $f\xi$ refers to

$$\lambda\xi_2, \dots, \xi_k \ (f\xi, \xi_2, \cdots, \xi_k),$$

and using the notation $f\xi$ implies that the type of ξ is the same as the type of the first input to f. I use this notation in a more relaxed way. The expression $f\xi$ refers to

$$\lambda\xi_1, \dots, \xi_{i-1}, \xi_{i+1}, \dots, \xi_k \ (f\xi_1, \dots, \xi_{i-1}, \xi, \xi_{i+1}, \dots, \xi_k)$$

where σ_i is the first type matching the type of ξ. Using the notation $f\xi$ implies that f has an input whose type is the type of ξ.

(2) Furthermore, when ";" appears in a list, it separates the higher tier inputs on the left from the lower tier inputs on the right. Thus, $f\overline{z};\overline{w};\overline{\alpha}$ is obtained by putting the tier 2 terms $\overline{z}$, the tier 1 terms $\overline{u}$, and the tier 0 terms $\overline{\alpha}$ into the matching inputs of f. Since all inputs have admissible type, $\overline{z}$ and $\overline{w}$ must be level 0. It must be emphasized that the use of ";" does not provide any additional structure beyond what is already present in a typed system; it simply allows one to abbreviate lengthy notation that would otherwise be required for specifying types.

The definition of $2T^+$ uses a generalized form of projection and composition. For projections π, one may have for example an equality such as $g = \pi_1 g$ in which the projected input, g, is level 1 instead of zero. This is formally accomplished by making the type of π_1 include the arguments to the projected input: if g is type $T_1 \to T_0$, say, then π_1 has type $[T_1 \to T_0, T_1, T_0]$.

For compositions, one uses a generalized form in which the the composed values may not be reduced to level 0: for example, if $h : (T_1 \to T_0) \to T_0$ and $g : T_2 \times T_1 \to T_0$ then we may compose them as $f \equiv \lambda x \ h(gx)$ where x has type T_2 and gx has type $T_1 \to T_0$. The defined f has type $T_2 \to T_0$. An instance of composition is *level 1* if the defined function is level 1. Together, generalized projection and composition can be used to define an apply functional: let $\overline{\sigma}$ be a level 1 type and let $k = \#\overline{\sigma} - 1$ and $\overline{\tau} = [\overline{\sigma}, \overline{\sigma}_1, \dots, \overline{\sigma}_k]$. Define

$$A_{\overline{\sigma}} \equiv \lambda r, \overline{n} \ \ \pi_{1,[\overline{\sigma}]}(\pi_{1,\overline{\tau}} r, \overline{n}), (\pi_{2,\overline{\tau}} r, \overline{n}), \dots, (\pi_{1+k,\overline{\tau}} r, \overline{n})$$

where r is level 1 type $\overline{\sigma}$. The operation A applies its first input to the second and subsequent inputs: $Ar, \overline{n} = r\overline{n}$. The type of $A_{\overline{\sigma}}$ is $[\overline{\sigma}, \overline{\sigma}_\circ]$.

Definition 2. $2T^+$ is the smallest set of operations with admissible input types defined from the constants and rules below.

- General projection: $\pi_{i,\overline{\sigma}}$, which produces the ith input, $1 \le i \le \#\overline{\sigma}$. Letting $\overline{\tau} \equiv \sigma_i$, the type of $\pi_{i,\overline{\sigma}}$ is $[\overline{\sigma}_\circ, \overline{\tau}_\circ]$.
- 0^{T_i} and 1^{T_i}, for $i \in \{0,1,2\}$. They are operations that produce the normal representations of 0 and 1.
- $*^{[T_j, T_i, T_j]}$ for $i > j \in \{0,1,2\}$; and also for $i = j = 2$. The result $m * n$ is the tree whose left child equals $m \in T_j$ and whose right child equals $n \in T_i$.

- $C^{[T_i,T_i,T_i,T_0,T_i]}$ for $i \in \{0,1,2\}$, defined by

$$Cn_0,n_1,n_2,n_3 = \begin{cases} n_0 \text{ if } n_3 = 0 \\ n_1 \text{ if } n_3 = 1 \\ n_2 \text{ otherwise} \end{cases}$$

- $L^{[T_i,T_i]}$ and $R^{[T_i,T_i]}$ for $i \in \{0,1,2\}$, defined by

$$\begin{array}{rr} L0 = 0 & R0 = 0 \\ L1 = 1 & R1 = 1 \\ L(n * m) = n & R(n * m) = m \end{array}$$

- General composition: If h of arity k is in the class, and $g_1, \ldots, g_k$ are in the class, then

$$f \equiv \lambda\overline{\xi} \;\; h(\overline{g}\overline{\xi})$$

is in the class. Here, if $[\sigma_1, \ldots, \sigma_k, \sigma_{k+1}]$ is the type of h then $g_i\overline{\xi}$ has type σ_i, possibly a level 1 type.
- Simultaneous 1-recursion: If $\overline{h}^0$, $\overline{h}^1$, and $\overline{h}^*$ are in the class, then so is $\overline{f}$ defined by:

$$\begin{aligned} f_l\ \overline{z}; 0, \overline{w}; \overline{\alpha} &= h_l^0\ \overline{z}; \overline{w}; \overline{\alpha} \\ f_l\ \overline{z}; 1, \overline{w}; \overline{\alpha} &= h_l^1\ \overline{z}; \overline{w}; \overline{\alpha} \\ f_l\ \overline{z}; (u * v), \overline{w}; \overline{\alpha} &= h_l^*\ \overline{z}; (u * v), \overline{w}; \overline{\alpha}, (\overline{f}\overline{z}; u, \overline{w}; \overline{\alpha}), (\overline{f}\overline{z}; v, \overline{w}; \overline{\alpha}) \end{aligned}$$

for all $\overline{z}$, u, v, $\overline{w}$, and $\overline{\alpha}$. The expressions on either side of these equations are level 0.
- Simultaneous 2-recursion: If $\overline{h}^0$, $\overline{h}^1$, and $\overline{h}^*$ are in the class, then so is $\overline{f}$ defined by:

$$\begin{aligned} f_l\ 0, \overline{z}; \overline{w}; \overline{\alpha} &= h_l^0\ \overline{z}; \overline{w}; \overline{\alpha} \\ f_l\ 1, \overline{z}; \overline{w}; \overline{\alpha} &= h_l^1\ \overline{z}; \overline{w}; \overline{\alpha} \\ f_l\ (x * y), \overline{z}; \overline{w}; \overline{\alpha} &= h_l^*\ (x * y), \overline{z}; \overline{w}; \overline{\alpha}, (\lambda\overline{w} \;\; \overline{f}x, \overline{z}; \overline{w}; \overline{\alpha}), (\lambda\overline{w} \;\; \overline{f}y, \overline{z}; \overline{w}; \overline{\alpha}) \end{aligned}$$

for all x, y, $\overline{z}$, $\overline{w}$, and $\overline{\alpha}$. The expressions on either side of these equations are level 0.

Rather than include a general rule of flat recurrence, I have just included the three important instances of the rule, namely C, L, and R.

Tier 2 terms can effectively be substituted into tier 1, and tier 1 terms can be substituted into tier 0, using an operation κ^σ defined by composing R and $*$. In [10], κ was defined by recursion.

In a system with infinitely many tiers, one should be able to admit recursion on all predicative function types and have $*$ at each tier having only one input from that tier.

3 Bounding $2T^+$

The first goal is to bound the output length of the functions by a polynomial.

For a tree u, the length $|u| \geq 1$ is the number of leaves in u, and the node height $\lceil u \rceil \geq 1$ is the number of nodes in a maximal root-to-leaf path. These can be generalized to functions by looking at the maximum output length.

Definition 3. For an operation g of admissible type, define a monotone function $|g|$ by: $|g|j,k \equiv \max_{\overline{w}} |g;\overline{w};|$, where the max is taken over $\{\overline{w} : \forall i\ |w_i| \leq j \wedge \lceil w_i \rceil \leq k\}$. Similarly, define monotone $\lceil g \rceil$ by $\lceil g \rceil j,k \equiv \max_{\overline{w}} \lceil g;\overline{w}; \rceil$ where the max is taken over the same values of $\overline{w}$.

In this notation, $|\overline{r}|j,k$ means $(|r_1|j,k),\ldots,(|r_{\#\overline{r}}|j,k)$.

The degree of the bounding polynomials is directly related to the nesting depth of recursion. Clote's definition of "rank" [6] can be modified to formalize the nesting depth of recursion:

Definition 4. The *2-rank* of f is ρ_2^f, and the *1-rank* is ρ_1^f, defined by:

- If $f \in 2T^+$ is one of the initial functions, then $\rho_2^f = \rho_1^f = 0$.
- If f is defined by composition from h and $\overline{g}$, then $\rho_2^f = \max\{\rho_2^h, \rho_2^{\overline{g}}\}$ and $\rho_1^f = \max\{\rho_1^h, \rho_1^{\overline{g}}\}$.
- If f is defined by simultaneous 2-recursion, then $\rho_2^f = \max\{1 + \rho_2^{h_i^*}, \rho_2^{h_i^0}, \rho_2^{h_i^1}\}_i$ and $\rho_1^f = \max\{\rho_1^{h_i^0}, \rho_1^{h_i^1}, \rho_1^{h_i^*}\}_i$.
- If f is defined by simultaneous 1-recursion, then $\rho_2^f = \max\{\rho_2^{h_i^0}, \rho_2^{h_i^1}, \rho_2^{h_i^*}\}_i$ and $\rho_1^f = \max\{1 + \rho_1^{h_i^*}, \rho_1^{h_i^0}, \rho_1^{h_i^1}\}_i$.

In many cases we will only be interested in recursions whose depth is not a constant. By examining the derivation of f one can determine whether or not the recursion variable of a particular recursion has been replaced by a constant — that is, by composition with a zero-ary function. I call these *constant recursions.*

Define $\hat{\rho}_2^f$ and $\hat{\rho}_1^f$ similarly to ρ_2^f and ρ_1^f, except that if f is defined by a constant 2-recursion or constant 1-recursion then $\hat{\rho}_2^f = \max\{\hat{\rho}_2^{h_i^0}, \hat{\rho}_2^{h_i^*}, \hat{\rho}_2^{h_i^*}\}_i$ and $\hat{\rho}_1^f = \max\{\hat{\rho}_1^{h_i^0}, \hat{\rho}_1^{h_i^1}, \hat{\rho}_1^{h_i^*}\}_i$.

Definition 5. Let $2T_{k,c}^+$ be the set of $f \in 2T^+$ such that $\hat{\rho}_2^f = k$ and $\hat{\rho}_1^f = c$.

Clearly, if $k' \leq k$ and $c' \leq c$ then $2T_{k',c'}^+ \subseteq 2T_{k,c}^+$, because one can always add extra recursions that have no effect on the computed function.

The definition of "nicely bounded", given below, is designed to suit the structure of $2T^+$. To get a rough idea of it, let's ignore various features of $2T^+$ such as the conditional function. In each $2T^+$ recursion, we build up a lopsided tree of height $\log n$. The deepest leaf of this tree is replaced with a tier 0 subtree. The other leaves are replaced by tier 1 or 2 subtrees. These higher-tier subtrees are added on by repeating $\log n$ applications of $*$; therefore they appear at depths decreasing from $\log n$ through 1. Each time we perform such a recursion, the constructed tree is tier 0, so outer nested recursions can only use this constructed tree once, as the deepest part of a larger tree. Overall, a typical constructed value is a lopsided tree in which a subtree of a tier 0 input appears at depth at most $O(\log^k n)$ and in which there are $O(\log^k n)$ tier 1 or 2 subtrees appearing at shallower and shallower depths.

Definition 6. An operation $\lambda\overline{z};\overline{w};\overline{a},\overline{r}\ \ f\overline{z};\overline{w};\overline{a},\overline{r}$ in $2T^+$ is *nicely bounded* if there is a constant $c_f \geq 1$ such that for all values $\overline{z};\overline{w};\overline{a},\overline{r}$:

- (δT_2) If f is tier 2,

$$\begin{aligned} |f\overline{z};\overline{w};\overline{a},\overline{r}| &\leq c_f \cdot \max\{1, |\overline{z}|\} \\ \lceil f\overline{z};\overline{w};\overline{a},\overline{r}\rceil &\leq \max\{\lceil\overline{z}\rceil\} + c_f \end{aligned}$$

- (δT_1) If f is tier 1,

$$\begin{aligned} |f\overline{z};\overline{w};\overline{a},\overline{r}| &\leq c_f \cdot \max\{1, |\overline{z}|, |\overline{w}|\} \\ \lceil f\overline{z};\overline{w};\overline{a},\overline{r}\rceil &\leq \max\{\lceil\overline{z}\rceil, \lceil\overline{w}\rceil\} + c_f \end{aligned}$$

- (δT_0) If f is tier 0, let

$$\begin{aligned} J_2 &= c_f \cdot \max\{1, \lceil\overline{z}\rceil\}^{\rho_2^f} \cdot \max\{1, |\overline{z}|\} \\ J_1 &= \max\{|\overline{w}|\} \\ J &= J_2 + J_1 \end{aligned}$$

$$\begin{aligned} K_2 &= c_f \cdot \max\{1, \lceil\overline{z}\rceil\}^{1+\rho_2^f} \\ K_1 &= \max\{\lceil\overline{w}\rceil\} \\ K &= K_2 + K_1 \end{aligned}$$

$$\begin{aligned} L &= \max\{\lceil\overline{z}\rceil\}^{\rho_2^f} \cdot K^{\rho_1^f} \cdot J \\ M &= \max\{\lceil\overline{z}\rceil\}^{\rho_2^f} \cdot K^{1+\rho_1^f} \end{aligned}$$

then

$$\begin{aligned} |f\overline{z};\overline{w};\overline{a},\overline{r}| &\leq L + \max\{|\overline{a}|, (|\overline{r}|J, K)\} \\ \lceil f\overline{z};\overline{w};\overline{a},\overline{r}\rceil &\leq M + \max\{\lceil\overline{a}\rceil, (\lceil\overline{r}\rceil J, K)\} \end{aligned}$$

It is evident that all of these bounding expressions are monotone. For level 2, monotonicity refers to the partial ordering on functions induced pointwise by the ordering of their level 0 inputs.

For the intuition behind (δT_0), consider that $(\lceil z\rceil^{\mu} + \lceil w\rceil)$ is approximately the height of a tier 1 value produced during the course of μ nested 2-recursions. These values can be used to define a tier 0 value using ρ_1 nested 1-recursions; the resulting height is approximiately $(\lceil z\rceil^{\mu} + \lceil w\rceil)^{\rho_1} + \lceil a\rceil$. On the other hand, the remaining $\rho_2 - \mu$ nested 2-recursions can be used to repeat this process. Due to the use of tiering, each repetition adds a height of $\lceil z\rceil^{\mu} + \lceil w\rceil$, giving a total height of $\lceil z\rceil^{\rho_2-\mu} \cdot (\lceil z\rceil^{\mu} + \lceil w\rceil)^{\rho_1} + \lceil a\rceil$. For every μ, this is bounded by $\lceil z\rceil^{\rho_2} \cdot (\lceil z\rceil^{\rho_2} + \lceil w\rceil)^{\rho_1} + \lceil a\rceil$. The expressions $|\overline{r}|J, K$ and $\lceil\overline{r}\rceil J, K$ are the length and height of tier 0 values produced by applying $\overline{r}$ to inputs bounded by J and K; these tier 0 values are treated in the same way as $\overline{a}$.

Theorem 7. *Every $f \in 2T^+$ is nicely bounded.*

The proof, which is omitted here, is by induction on the derivation of f. Although the concepts used in the proof are straightforward, involving nothing more complicated than monotonicity of type 2 functionals, the proof is quite intricate due to the large number of cases and the nested expressions in the definition of "nicely bounded".

4 Simulating ATM computations

In this section I would like to show that $2T^+$ is as powerful as alternating Turing machines (ATMs). Ruzzo proved that the computation graph of an ATM is essentially the same as a uniform circuit [16].

Computation graphs of ATMs have previously been used by Bloch to validate one direction of a recursive characterization of NC^1 [4]. A generalization to polylog depth was achieved by an encoding of subcomputations that generated superpolynomially large output values; this resulted in a characterization of polylog depth superpolynomial size circuits [4].

Definition 8. The *left height*, $\lceil x|$, of a tree x is the number of nodes in the leftmost branch of x. Formally, $\lceil 0| = \lceil 1| = 1$ and $\lceil (x * y)| = 1 + \lceil x|$. Similarly, the right height is $|0\rceil = |1\rceil = 1$ and $|(x * y)\rceil = 1 + |y\rceil$. A *path* is a tree that is either 0, 1, or is $p * 0$ or $p * 1$ for some path p. Given a vector of trees $\overline{x}$, the *expansion* of $\overline{x}$ is the tree $X_{\overline{x}}$ defined by: X_{x_1} is x_1; and $X_{x_1,\ldots,x_{k+1}}$ is defined by

$$\begin{aligned} X_{x_1,\ldots,x_k,0} &= X_{x_1,\ldots,x_k} \\ X_{x_1,\ldots,x_k,1} &= X_{x_1,\ldots,x_k} \\ X_{x_1,\ldots,x_k,y*z} &= X_{x_1,\ldots,x_k} \text{ with each leaf replaced by } X_{x_1,\ldots,x_k,y}. \end{aligned}$$

An equivalent definition is: $X_{x_1,\ldots,x_{k+1}}$ is obtained by forming $\lceil x_{k+1}|$ layers, the top layer consisting of $X_{x_1,\ldots,x_k}$ and each succeeding layer being obtained by replacing the leaves of the preceding layer with copies of $X_{x_1,\ldots,x_k}$.

If we have $\#\overline{x} = k$ and each x_i is a B presentation (i.e. a balanced tree) with $|x_i| = n$, then $\lceil x_i|$ is approximately $\log n$, and $\lceil X_{\overline{x}}\rceil$ is approximately $\log^{\#\overline{x}} n$. Correspondingly, $|X_{\overline{x}}|$ is approximately $2^{\log^{\#\overline{x}} n}$, a superpolynomial size.

$2T^+$ can effectively recurse over the expansion of $\overline{x}$ by using a nested recursion for each x_i. It is achieved by defining a function $E^h_k\overline{x}; p, v; f$ where $\#\overline{x} = k$. The superscript h indicates the operation that is applied at each interior node of $X_{\overline{x}}$. The argument f is an operation defining the value at each leaf of $X_{\overline{x}}$. The argument p is the path that was followed to reach the current node in $X_{\overline{x}}$ (i.e. to reach the root of $X_{\overline{x}}$). At the root of $X_{\overline{x}}$, the path p may be nonempty, to indicate the location of $X_{\overline{x}}$ in a larger tree, such as in $X_{\overline{x},z}$. When we reach a leaf, for example, the path p is available for use by the function f. The argument v is a parameter value that is given unchanged as input to f and h. The value of $E^h_k\overline{x};p;f$ is the result of extending the path p to reach a leaf of $X_{\overline{x}}$, applying f at that leaf, and then applying h at each interior node of $X_{\overline{x}}$.

$$\begin{aligned}
&E^h_1 0;p,v;f &&= f;p,v;\\
&E^h_1 1;p,v;f &&= f;p,v;\\
&E^h_1(x*y);p,v;f &&= h;p,v;(E^h_1 x;p*0,v;f),(E^h_1 y;p*1,v;f)
\end{aligned}$$

$$\begin{aligned}
&E^h_{k+1}\overline{z},0;p,v;f &&= E^h_k\overline{z};p,v;f\\
&E^h_{k+1}\overline{z},1;p,v;f &&= E^h_k\overline{z};p,v;f\\
&E^h_{k+1}\overline{z},(x*y);p,v;f &&= E^h_k\overline{z};p,v;(\lambda p\ \ E^h_{k+1}\overline{z},x;p,v;f)
\end{aligned}$$

The definition of E^h_{k+1} is analogous to the definition of $X_{x_1,\ldots,x_k,x_{k+1}}$. Note that $\rho^E_1 = \max\{\rho^f_1,\rho^h_1\}$ and $\rho^{E_k}_2 = k$.

Observe that the parameters v must be in T_1 according to the type scheme being used for the development. If these parameters were level 1, then f would be level 2 and E would be level 3.

The tree $X_{\overline{x}}$ can be used as a computation tree for a polylog time alternating Turing machine. To do so, we must allow constant recursions to allow for constants in the computation time of the ATM. When $K > k$ below, we will have $\rho > \hat{\rho}$ (definitions are in section 3).

Definition 9. An ASPACETIME(P,k) machine is an alternating Turing machine, M, with the following features.

The resources used by M are determined by a presentation system P and an integer k. The computation graph of M is described by a function $\overline{x} : N \to (T_2)^K$ such that $K \geq k$, and such that for exactly k value of $i \in \{1,\ldots,K\}$ one has $\forall \mathbf{x}\ \ \overline{x}_i = P(\mathbf{x})$; and $\overline{x}_i$ is a constant for all the other values of $i \in \{1,\ldots,K\}$. The computation tree on input $\mathbf{x}$ is obtained from $X_{\overline{x}}$ by labeling the interior nodes with either universal or existential states according as the depth of the node from the root is even or odd, respectively; and by labeling each leaf with T, F, L, or ¬L. Finally, M has a constant number of bidirectional read/write tapes. The number of cells used is only bounded by the running time, $O(\lceil X_{\overline{x}} \rceil)$.

The value returned by M at a leaf labeled T is 1; at a leaf labeled F it is 0; at a leaf labeled L it is $(\mathbf{x}/2^i) \bmod 2$ where i is the value to the left of the head on tape number 1; and at a leaf labeled ¬L it is $1 - ((\mathbf{x}/2^1) \bmod 2)$ where i is as before. Each transition of the machine is allowed to read, then write, one bit on each of the tapes.

Theorem 10. *For $k \geq 1$, a function is computable by an $O(\log^k n)$ depth, $2^{O(\log^k n)}$ size U_{E^*}-uniform circuit iff it is computed by an* ASPACETIME(B,k) *machine. For $k \geq 2$, if a function is computable by an $O(n^{k-1})$ depth, $2^{O(n^{k-1})}$ size U_{E^*}-uniform circuit then it is computed by an* ASPACETIME(S,k) *machine.*

Proof. (Sketch). See Ruzzo [16] for a proof that uniform depth T size 2^S is equivalent to alternating Turing machine time $O(T)$ and space $O(S)$, provided that $S = \Omega(\log n)$ and $T = \Omega(\log n)$ are suitably constructable.

The class of $\textsc{aspacetime}(B, k)$ machines is equivalent to the class of alternating Turing machines using space logarithmic in $|\mathrm{x}|$ and at most $\lceil B(\mathrm{x}) \rceil^k$ alternations. Requiring the states to alternate on each step, requiring exactly two successor states, and requiring the input to be accessed only at the leaves, are well known to increase the depth of an $\Omega(\log^k n)$-depth machine ($k \geq 1$) by at most a constant factor. The size of the computation tree can be increased by a constant amount by using $K > k$ — additive constant factors can be absorbed by multiplicative constant factors because $\lceil \mathrm{x} \rceil \geq 1$. Although the shape of the computation tree is not perfectly balanced if $|\mathrm{x}|$ is not a power of 2, this also can be absorbed by increasing the depth by a constant factor: the longest and shortest root-to-leaf path lengths in $B(\mathrm{x})$ differ by at most 1, and therefore the imbalance of the expanded tree is at most a difference of $\lceil \mathrm{x} \rceil^k$ nodes between the lengths of the shortest and the longest paths.

For the S presentation system, the situation is analogous. Although the tree is more unbalanced, the shortest paths in the expansion $X_{\overline{x}}$ are still length $\Omega(n^{k-1})$. Notice that S presentations and the expansion $X_{\overline{x}}$ coincide in favoring the left leg of the presentation. □

Lemma 11. *Using simultaneous 1-recursion, one can define a function* ($\textsc{bit}; v; i$) *that returns the* $\{i\}$*th bit of* $\{v\}$*, provided that* i *is a path and* v *is a* B *or* S *presentation with* $\{i\} \leq |v|$. *In fact, the definition has* $\rho_1^{\mathrm{BIT}} = 1$.

The main complication of a proof for this lemma is that not all root-to-leaf paths of a B presentation are the same length, and therefore i does not provide a very direct way to determine the bits of the path.. A detailed proof is omitted.

Theorem 12. *Let* P *be either* B *or* S. *For every* $\textsc{aspacetime}(P, k)$ *machine there is an operation* $f \in 2T^+_{k,1}$ *such that for all* $\mathrm{x} \in N$, *the output of the machine on input* x *is given by* $\{fP(\mathrm{x})\}$.

Proof. The proof is a direct application of the function E.

Each possible machine state is encoded by a constant in T_0, and each tape state is encoded by two paths $a \in T_0$ and $b \in T_0$, such that a contains the bits at or to the left of the head and b contains the reverse of the bits to the right of the head. Thus the machine configuration consists of a vector of tier 0 paths; let $\overline{\mathbf{0}}$ be the initial machine configuration. A constant number of the low-order bits of these paths can be manipulated in a suitable way using functions such as $a * 0$, $a * 1$, and $L;; a$. Specifically we can define functions $\overline{\Delta}^0$ and $\overline{\Delta}^1$ such that $\Delta^i_l;; \overline{a}$ is the lth component of the configuration obtained by choosing the branch $i \in \{0, 1\}$ from configuration $\overline{a}$. Now using $\overline{\Delta}^0$ and $\overline{\Delta}^1$ in the step functions, simultaneous 1-recursion can be used to define functions $(\Delta^*_l; p; \overline{a})$ giving the lth component of the configuration that results from configuration $\overline{a}$ when path p is followed in the computation tree.

The leaf-value function $f; p, v;$ now can be defined by obtaining $(\overline{\Delta}^*; p; \overline{\mathbf{0}})$ and testing the state for $\textsc{t}$, $\textsc{f}$, $\textsc{l}$ and $\neg\textsc{l}$; in the latter cases $\textsc{bit}$ is used to obtain the value from v. When i is a path represents the contents of tape number 1 (up to the tape head) at a leaf L or ¬L, and v is a copy of the original input $B(\mathbf{x})$, the value of (BIT; v; i) is the relevant bit of $\mathbf{x}$.

The interior function h is defined by $h;p;a,b =$ "if (parity p) $= 0$ then (a AND b) else (a OR b)". Parity is easily defined by 1- recursion, and AND and OR are defined using the conditional.

Let $K \geq k$ and $\overline{x}$ be as in the definition of an ASPACETIME(P, k) machine: $\overline{x}$ is a vector of $K - k$ constants and k copies of $B(\mathbf{x})$ such that the expansion of $\overline{x}$ is the computation graph of M on input $\mathbf{x}$. Letting i satisfy $\overline{x}_i = B(\mathbf{x})$, the ASPACETIME output is $E^h_K\overline{x}; p, x_i; f$.

The ρ_2 for the defined function is $\rho_2^{E_K}$, which is K. Its 1-rank is the 1-rank of f. Since f is a composition of the initial functions with BIT and $\overline{\Delta}^*$, one has $\rho_1^f = \max\{\rho_1^{\mathrm{BIT}}, \rho_1^{\overline{\Delta}^*}\} = 1$.

Considering that all but k of the K recursions defining E_K are substituted by the constant values among $\overline{x}$, one has that $\hat{\rho}_2$ for the defined function is at most k. □

This establishes one direction of the main theorems.

Theorem 13. *Uniform polylog parallel time is contained in the level 1 section of $B/2T^+$. In fact, for all $k \geq 1$, uniform parallel time $O(\log^k n)$ is contained in the level 1 section of $B/2T^+_{k,1}$.*

Theorem 14. *Uniform polynomial parallel time is contained in the level 1 section of $S/2T^+$. In fact, for all $k \geq 2$, uniform parallel time $O(n^k)$ is contained in the level 1 section of $S/2T^+_{k+1,1}$.*

Proof. These two corollaries are a direct consequence of the preceding simulation theorem together with the modified form of Ruzzo's theorem. Notice that the preceding theorem does not require any bound on the space used by the ATM. □

5 Simulating $2T^+$ operations

The second direction to be proved is that $2T^+$ operations can be computed by ATMs within suitable resource bounds.

Theorem 15. *The level 1 section of $B/2T^+$ is contained in uniform polylog parallel time. In fact, for $k \geq 1$, the level 1 section of $B/2T^+_{k,1}$ is contained in uniform parallel time $O(\log^{k+2} n)$.*

Theorem 16. *The level 1 section of $S/2T^+$ is contained in uniform polynomial parallel time.*

The use of circuits for a formal proof of these theorems would be awkward because the uniformity conditions would require a mapping from the implicitly uniform structure of $2T^+$ derivations to an explicit machine computing, say, the extended connection language of the circuit family. However, in other respects circuits provide a direct way to understand $B/2T^+$ operations. Therefore, let me use circuits to give an informal idea of the result.

Consider circuits with many output bits and with oracle gates for the level 1 inputs. Each oracle gate is labeled with the name of one of the level 1 inputs, and has any number of bits input to it and a corresponding number of bits output from it. Furthermore, we maintain the invariant that each root-to-leaf path passes through at most one oracle gate. This invariant corresponds to the fact that the types of level 1 inputs are predicative: the result of applying a level 1 input cannot be used, either directly or indirectly, as the input to another level 1 oracle application. An exception occurs when C is used, as in $(h; u, v, w; f, g) = (f; (C; u, v, w; (gu));)$ — here the T_0 output of gu is being used indirectly as an input to f. Such a usage, however, can always be rewritten as in $(h; u, v, w; f, g) = C; (fu), (fv), (fw); (gu)$. Intuitively, the presence of at most one oracle gate along any given path allows us to unroll 2- recursions as iterations. For unrolling 1-recursions one similarly requires that each literal testing a tier 0 input cannot be used indirectly as an input to an oracle gate. For example, level 0 oracles testing input bits are not descendants of level 1 oracle gates.

To show that there is at most one oracle gate along each root to leaf path, one assigns a tier to each gate of the circuit: the tier of a leaf is the tier of the input being tested at that leaf, or is 1 if the leaf is a constant; the tier of an AND or OR gate is the minimum of the tiers of its inputs; and the tier of a level 1 oracle is 0 (since the oracle function has admissible type with output in T_0). The invariant is that the gates input to the level 1 oracles are all tier 1; this corresponds to the fact that admissible level 1 oracles have all tier 1 inputs.

In the actual circuit corresponding to a $2T^+$ operation, the number of inputs to the oracle gates is bounded by the size of the circuit, and this is bounded using the Bounding Lemma. The size of the circuit is not, naively, 2^l where l is the length of computed values, because many of the bits in a computed value are simply copies of the bits of the input.

It is easy to construct circuits for the initial functions. For 1-recursion, one replaces the leaves of the circuit for h^* with copies of the circuit for h^*. The substituted leaves are oracles for the level 0 input bits. The substitution is repeated $\lceil v \rceil - 1$ times, where v is the input being recursed on. Finally, one replaces the resulting leaves with a circuit that uses the input bit at that leaf to select the output from either a copy of h^0 or a copy of h^1. The resulting circuit depth is $\lceil v \rceil$ times what it used to be; for the B presentaton $\lceil v \rceil = O(\log n)$. Although at first it might appears that this construction could generate $\lceil v \rceil$ oracle gates along a root-to-leaf path, in fact this does not happen because the leaves being replaced correspond to tier 0 inputs, and these will not be the descendants of oracle gates.

For 2-recursion, start with a circuit for h^*: it is parameterized by the bits for the recursion variable x, it has height $\log^{k-1} n$, and it contains oracle gates for accessing the level 1 inputs. To get a circuit corresponding to the recursion, replace each oracle gate for the critical terms with a copy of the circuit for h^*, but parameterized by either Lx or Rx. This substitution is repeated $\lceil x \rceil - 1$

times. At the end, there is only one parameter bit of x given as input to the oracle gate. (Of course the oracle gate can have other input bits corresponding to non-recursion inputs). Then, replace these oracle gates with a circuit that uses the parameter bit x to select the output from either a copy of h^0 or a copy of h^1. By placing the copies of h^0 and h^1 in parallel, one preserves the invariant that each root-to-leaf path contains at most one oracle gate. It is easy to see that this invariant is also maintained by the repeated substitution of h^*. Because there was originally at most one oracle gate along each path, the height of the resulting circuit is at most $(\lceil x\rceil - 1) \cdot \lceil h^*\rceil + \max\{\lceil h^0\rceil, \lceil h^1\rceil\}$. The size of the circuit does increase rapidly up to $n^{O(\log^k n)}$ due to the presence of multiple oracle gates for the critical terms.

In the case of $S/2T^+$, one can follow exactly the same proof, again obtaining bounds on the height of the circuit in terms of $\lceil x\rceil$. The condition that there is at most one oracle gate along each root-to-leaf path is now considerably more severe relative to the size of the circuit: a typical root-to-leaf path now passes through a much larger number of nodes.

6 Conclusion

The results of this paper confirm a direct relationship between type 2 recursion and parallel computation. It has been shown that in defining a recursion class, the output length can be treated separately from the complexity of the bits in the output. This suggests that output length is not a barrier to obtaining natural recursive characterizations of parallel complexity classes. Furthermore, the results provide additional evidence that the power of type 2 recursion can be sharply curtailed by appropriately ramified structures.

The reduction in output length was achieved without reducing the space available to the corresponding ATM or circuit computation. If the ATM space were $O(\log n)$, a characterization of NC would be obtained. Thus an evident challenge for future research is to show how to characterize lower complexity classes, such as NC, in a similar way.

The definition of $2T^+$ has a coincidence between the tier of the recursion variable and the type level of the recursion allowed on this variable. At the same time, Leivant [10] has shown that two tiers are as good as infinitely many for characterizing polytime with type 1 recursions; and Leivant and Marion [12] have shown that three tiers are as good as infinitely many for characterizing polyspace with type 2 recursions. This makes one wonder whether the use of higher tiers has some intrinsic relationship to the use of higher types.

The formal use of presentations such as B and S seems, to this author, connected to the question of domain ordering from finite model complexity theory [8]. Intuitively, any recursively defined presentation system has implicit in it an ordering, i.e. the ordering given by the recursive structure of the presentations. Thus, the thesis that various complexity classes can be naturally obtained by considering different presentations for a single class of operations, seems related to the thesis that various ordering relations will induce different complexity classes under a fixed iterated first-order language. We cannot rule out the possibility that a single, natural, recursion class C could be defined such that many, if not all, natural complexity classes are induced by various presentation systems for C.

7 Acknowledgments

I would like to thank Daniel Leivant for his helpful comments, and for sending copies of his papers. Thanks to Stephen Cook for helping set me straight on some important points. Finally I would like to thank the University of Toronto Department of Computer Science for the use of their facilities in producing this work.

References

1. S. Bellantoni and S. Cook, "A New Recursion-Theoretic Characterization of the Polytime Functions", *computational complexity* v. 2, p. 97-110, 1992. Extended Abstract appeared in *Proc. 24th Symposium on the Theory Of Computing*, 1992.
2. S. Bellantoni, "Predicative Recursion and Computational Complexity", Ph.D. Thesis, Department of Computer Science, University of Toronto, 1992. Available as Technical Report 264/92.
3. S. Bellantoni, "Further complexity characterizations using predicative recursion", submitted for publication.
4. S. Bloch, "Functional Characterizations of Uniform Log-depth and Polylog-depth Circuit Families", in *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, IEEE (1992).
5. A. Chandra, D. Kozen, L. Stockmeyer, "Alternation", in *Journal of the Association for Computing Machinery*, v. 28, n. 1, p. 114- 133, Jan 1981.
6. P. Clote, "Sequential, machine-independent characterizations of the parallel complexity classes *ALogTime*, AC^k, NC^k, and *NC*", in *MSI Workshop on Feasible Mathematics*, Birkhauser, 1989.
7. A. Cobham, "The intrinsic computational difficulty of functions". In Y. Bar-Hillel ed., *Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science*, p. 24-30. North Holland, Amsterdam, 1964.
8. N. Immerman, "Languages That Capture Complexity Classes", *SIAM Journal of Computing*, p. 760-778, v. 16 (1987).
9. D. Leivant, "Subrecursion and lambda representation over free algebras (Preliminary summary)", in *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhauser 1990.
10. D. Leivant, "Ramified recurrence and computational complexity I: Word recurrence and poly-time", in *Feasible Mathematics II*, P. Clote and J. B. Remmell, eds., Birkhauser, 1995.
11. D. Leivant, "Ramified recurrence and computational complexity III: Higher type recurrence and elementary complexity". Preliminary version appears in Logic From Computer Science 1994, Nerode and Matyasevich, eds., Lecture Notes in Computer Science, Springer Verlag.

12. D. Leivant, J. Y. Marion, "Ramified recurrence and computational complexity II: Substitution and poly-space", in *Computer Science Logic*, J. Tiuryn and L. Pacholsky, eds., Lecture Notes in Computer Science, Springer Verlag.
13. D. Leivant, J. Y. Marion, "Applicative characterization of poly- space (extended summary)", Manuscript, 1994.
14. A. Nguyen, *A Formal System for Linear-Space Reasoning*, M.Sc. Thesis, Department of Computer Science, University of Toronto, 1993.
15. J. Otto, *Tensor and Linear Time*, Logic and Computational Complexity, D. Leivant, ed., Lecture Notes in Computer Science, Springer Verlag, 1995.
16. W. Ruzzo, "On Uniform Circuit Complexity", Journal of Computer and System Sciences, v. 22, p. 365-383, 1981.
17. H. Simmons, "The Realm of Primitive Recursion", *Archive for Mathematical Logic*, v. 27, p. 177+, Springer Verlag, 1988.

Type 2 Polynomial Hierarchies

Anil Seth

The Institute of Mathematical Sciences
C.I.T. Campus, Taramani
Madras 600113, India.
e-mail: seth@imsc.ernet.in

Abstract. In this paper we examine type 2 analogs of the type 1 polynomial hierarchy and show some limitations on finding a completely faithful type 2 analog. We survey most of the notions of type 2 poly-hierarchies already proposed in the literature and present two natural definitions of type 2 poly-hierarchies. We also introduce various resource bounded reductions between functionals of type 2.

1 Introduction

Functionals of type 2 are total functions which take type 1 functions and natural numbers as arguments and output a natural number. In recent years some generalizations of type 1 poly-time to higher types have been studied. Many important classes other than poly-time are also studied in type 1 complexity theory. One example is polynomial hierarchy, [13]. In this paper we investigate possible generalizations of type 1 polynomial hierarchy to type 2.

Some possible generalizations are defined and studied in [14]. However there are several definitions of type 2 polynomial hierarchies in the literature and it is not clear which is the right one. Many questions about the properties and interrelationships of these proposed hierarchies are open, see [14]. In this paper we try to investigate the issue systematically by writing down some natural conditions for type 2 polynomial hierarchy, in the same way as conditions 8.1-8.3 in [8] are proposed for any class of feasible functionals. Surprisingly we find that these natural conditions lead to contradiction. Therefore there can be no type 2 polynomial hierarchy which possesses all these properties, nevertheless some attractive generalizations are possible. We survey the previous definitions in this light and present some new results about them. We later present two possible definitions of type 2 polynomial hierarchies, each of which is natural from a different viewpoint. These definitions have appeared in literature in slightly variant forms. These notions also arise naturally in the context of type 2 feasible functionals definable in various systems of bounded arithmetic which we study elsewhere.

The rest of this paper is organised as follows. In section 2 we show limitations on finding a faithful analog of type-1 poly-hierarchy in type 2. In section 3 we review various definitions proposed in the literature for type 2 poly-hierarchies. In the next section we prove some results, about these proposed classes left open in the literature. In section 5 we extend the various notions of time bounded reducibilities between type-1 functions to analogous reductions between type 2 functionals. These notions of reductions are shown to be different on the space of computable functionals. In section 6 we examine in some detail a notion of type 2 poly-hierarchy which is used by V. Harnik in [9] to generalize some results of [5]. In section 7 we investigate another natural notion of type 2 poly-hierarchy and prove some of its properties. The next section relates this poly-hierarchy with a class already considered in the literature. Finally, section 9 contains some concluding remarks. For the terminology and notation not defined in this paper we refer the reader to [10].

2 Desirable conditions and limitations for type 2 polynomial hierarchy

In this section we propose some desirable conditions for a type 2 polynomial hierarchy, in the same spirit as the necessary conditions proposed in [8] for a class of feasible functionals. These conditions are a straightforward lifting of some characteristic properties of the type 1 polynomial hierarchy.

Let $\Sigma^p_{2,k}, \Pi^p_{2,k}$ denote the classes of type 2 predicates of k^{th} level of the proposed polynomial hierarchy ($\Pi^p_{2,k}$ consists of complement of predicates in $\Sigma^p_{2,k}$) and let $\Box^p_{2,k}$ be the class of type 2 functionals of k^{th} level of the proposed polynomial hierarchy. Σ^p_k, Π^p_k denote the predicates in the k^{th} level of the type 1 polynomial hierarchy. $\Box^p_k$ denotes functions of k^{th} level of this hierarchy. More specifically, Σ^p_0, Π^p_0 denote polytime computable predicates and $\Box^p_0$ denotes poly-time functions, $\Box^p_k$ denotes functions computable in polytime using Σ^p_k oracle. (This definition of $\Box^p_k$ is slightly different from the one given in [2]).

The following are expected conditions for a type 2 polynomial hierarchy.

1. $\Sigma^p_{2,0} = \Pi^p_{2,0} =$ 0,1 valued functionals in $\Box^p_{2,0}$. The class $\Box^p_{2,0}$ satisfies the conditions laid down for a class of feasible functionals.
2. $\Box^p_{2,k} = \Box^p_{2,0}(\Sigma^p_{2,k})$ (= λ closure of $\Box^p_{2,0}$ and $\Sigma^p_{2,k}$).
   From this it follows that the class $\Box^p_{2,k}$ is closed under λ abstraction and application.
3. All type 1 functions constructed from the functionals in $\Box^p_{2,k}$, type 1 functions in $\Box^p_k$ and λ calculus operations are in $\Box^p_k$.
4. Each $\Sigma^p_{2,k}$ contains 0-1 valued functionals in $\Box^p_{2,k}$ and each $\Sigma^p_{2,k}$ is closed under $\Box^p_{2,0}$ functional bounded existential quantifier in natural number variable, that is, a quantifier of the type $\exists|x| \leq |\phi|$ where x is a natural number variable and ϕ is a functional in $\Box^p_{2,0}$. Similarly each $\Pi^p_{2,k}$ is closed under the corresponding universal quantifier.

Theorem 1 *Existence of a type 2 polynomial hierarchy with properties (1)-(4) above implies the collapse of type 1 polynomial hierarchy.*

Proof: Let $A(z) = \exists|x| \leq p(|z|)\forall|y| \leq p(max\{|z|,|x|\})[B(x,y,z)]$, where p is a polynomial and $B(x,y,z)$ a poly-time predicate, be a Σ^p_2 complete predicate.

Consider the type 2 predicate $F(f,z)$ defined as $\exists|x| \leq p(|z|)[f(z,x) = 1]$

F is in $\Sigma^p_{2,1}$ level of any type 2 polynomial hierarchy that satisfies conditions (1)-(4) laid down above. (By (4), as predicate $P(f,z,x) \equiv [f(z,x) = 1] \in \Box^p_{2,0}$)

Let $\underline{f}(z,x)$ be the characteristic function of predicate $\forall|y| \leq p(max\{|z|,|x|\})[B(x,y,z)]$.

Therefore $\underline{f} \in \Box^p_1$.

Now the type 1 predicate

$F(\underline{f},z)$

$= \exists|x| \leq p(|z|)[\underline{f}(z,x) = 1]$

$= \exists|x| \leq p(|z|)[\forall|y| \leq p(max\{|z|,|x|\})[B(x,y,z)]]$

$= A(z)$

Since $A(z)$ can be constructed using $\Sigma^p_{2,1}$ functional, $\Box^p_1$ function and substitution and since $\Sigma^p_{2,1}$ preserves $\Box^p_1$, we have that $\Sigma^p_2 \subseteq \Box^p_1$. This implies $\Sigma^p_3 = \Sigma^p_2$. □

In view of this theorem we have to give up the search for a type 2 polynomial hierarchy with all the above properties. However the notion of type 2 polynomial hierarchy has still been useful (see [9]), so we shall consider the hierarchies that satisfy only a reasonable subset of the above conditions.

3 Previous approaches

In this section we recall various notions of type 2 polynomial hierarchies that have been proposed so far in the literature. As shown in the previous section each of these hierarchies must fail to satisfy at least one property from (1)-(4). In the next section we study the properties of these hierarchies in detail and will answer several open question about them. We recall the following definitions from [14].

Type 2 relations are 0,1 valued functionals of type 2. Class BFF [5] is referred to as class **POLY** in [14], we shall therefore use the two terms interchangeably in this paper.

Definition 1 *The class of polynomially bounded relations is defined as the least class of predicates which contains 0-1 valued basic feasible functionals and is closed under length bounded quantifications of the form* $\exists|x| \leq |\phi(\mathbf{f},\mathbf{y})|$ *and* $\forall|x| \leq |\phi(\mathbf{f},\mathbf{y})|$ *, where* $\phi \in BFF$.

Townsend [14], classified polynomially bounded relations in a hierarchy as follows. In order to keep the notation distinct we shall describe levels of this hierarchy using boldface.

Definition 2 $\boldsymbol{\Sigma}_{\mathbf{0}}^{\mathbf{p}} = \boldsymbol{\Pi}_{\mathbf{0}}^{\mathbf{p}}$ *is the class of relations in BFF.*

$\boldsymbol{\Sigma}_{\mathbf{n+1}}^{\mathbf{p}}$ *is the class of relations definable by prefixing a* $\exists |x| \leq |\phi(\mathbf{f}, \mathbf{y})|$ *quantifier to relations in* $\boldsymbol{\Pi}_{\mathbf{n}}^{\mathbf{p}}$.

$\boldsymbol{\Pi}_{\mathbf{n+1}}^{\mathbf{p}}$ *is the class of relations definable by prefixing a* $\forall |x| \leq |\phi(\mathbf{f}, \mathbf{y})|$ *quantifier to relations in* $\boldsymbol{\Sigma}_{\mathbf{n}}^{\mathbf{p}}$.

(here $\phi \in BFF$*)*

It is left open in [14] to answer if the above hierarchy is proper. It is also conjectured in [14] that Σ levels of the above hierarchy are not closed under length bounded existential quantification. We will answer both these questions in the next section.

Definition 3 *The relativized class* $\Sigma_n^{p,g}$*; $n \geq 0$ is defined as follows.* $\Sigma_n^{p,g} = \{F(\mathbf{f}, \mathbf{y}) |$ *there is some* $G \in \Sigma_n^p$ *and* $F(\mathbf{f}, \mathbf{y}) = G(\mathbf{f}, g, \mathbf{y})\}$. Π_n^p *is defined similarly.*

Victor Harnik has defined a similar hierarchy by taking g to be type 1 poly- hierarchy functions of appropriate level and $G \in BFF$ in the above definition. We shall study that hierarchy in section 6.

In [14], yet another hierarchy is defined as follows.

Definition 4 $\underline{\boldsymbol{\Sigma}}_{\mathbf{n}}^{\mathbf{p}} = \cup_g \Sigma_n^{p,g}$.

$\underline{\boldsymbol{\Pi}}_{\mathbf{n}}^{\mathbf{p}} = \cup_g \Pi_n^{p,g}$. $n \geq 0$

In the next section we will show that this hierarchy collapses to the first level. This answers another question of [14].

4 New results on the previous work

In this section we investigate the properties of the hierarchies defined in [14]. These results will also be useful in the later sections of this paper.

4.1 Separation of type 2 classes

The following proof shows the separation among levels of the hierarchy of polynomially bounded relations, however the same idea can be used to show separation of any two classes for which analogous type 1 classes are different under some oracle. The idea here is to regard the oracle as type 1 input to a functional. It has been observed independently in the literature that oracle separation can be used to show separation for type-2 classes though we have not seen a detailed argument. We present full details here so that it becomes clear exactly when oracle separation can be converted into an actual type 2 separation. To use it in the later sections we just notice that all conditions for this argument to be applicable are fulfilled.

In keeping with the terminology of [14] we use lightface symbols to denote type 1 complexity classes and boldface symbols to denote analogous type 2 classes.

Lemma 1 : $\boldsymbol{\Sigma}_{\mathbf{n}}^{\mathbf{p}} \subset \boldsymbol{\Sigma}_{\mathbf{n+1}}^{\mathbf{p}}$.

($\subset$ stands for a *proper* subset.)

Proof: Note that, Σ_n^p denotes the levels of the type 1 polynomial hierarchy and $\boldsymbol{\Sigma}_{\mathbf{n}}^{\mathbf{p}}$ denotes the levels of the hierarchy of polynomially bounded relations.

Let g be a 0,1 function such that $\Sigma_n^{p,g} \subset \Sigma_{n+1}^{p,g}$, (from [15], such a function exists).

Let $f(x)$ be a 0,1 function $\in \Sigma_{n+1}^{p,g} - \Sigma_n^{p,g}$.

By the characterization of polynomial hierarchy we have that

$[f(x)=1] \equiv [\exists |y_{n+1}| \leq |x|^i \forall |y_n| \leq |x|^i \ldots Q|y_1| \leq |x|^i R^g(x, y_1, \ldots y_{n+1})]$.

Q is $\exists$ if n is odd, otherwise Q is $\forall$. R is a PTIME predicate computable in $O(m^i)$ time relative to oracle g.

Define functional $G(x, y_1, \ldots, y_{n+1}, h)$ as follows. The machine computing G on input $x, y_1, \ldots y_{n+1}, h$ simulates R on $x, y_1, \ldots y_{n+1}$ with oracle h. If an answer > 1 is returned by oracle or the computation time exceeds $|max\{x, y_1, \ldots y_{n+1}\}|^i$ then the machine stops. So G is in **POLY**.

Let $F(h,x) = \exists |y_{n+1}| \leq |x|^i \forall |y_n| \leq |x|^i \ldots Q|y_1| \leq |x|^i G(x, y_1, \ldots y_{n+1}, h)$

Q is $\exists$ if n is odd, otherwise Q is $\forall$. Observe that $F(g,x) = f(x)$.

$F \in \Sigma^{\mathbf{P}}_{\mathbf{n+1}}$. We claim that $F \notin \Sigma^{\mathbf{P}}_{\mathbf{n}}$.

To see this assume towards a contradiction that $F \in \Sigma^{\mathbf{P}}_{\mathbf{n}}$,

Now $F(h,x) = \exists |y_{n+1}| \leq |s_{n+1}(x,h)| \forall |y_n| \leq |s_n(x, y_{n+1}, h)| \ldots$
$\ldots Q|y_1| \leq |s_1(x, y_{n+1}, y_n, \ldots, y_2, h)| T(x, y_1, \ldots, y_{n+1}, h)$,

where each $s_1, s_2, \ldots s_{n+1}, T$ are computable by $O(m^k)$ POTMs (In fact they are in **POLY**).

(On 0,1 valued functions h, $|s_i(x, y_{n+1}, \ldots, y_{i+1}, h)|$ is bounded by a polynomial of degree k in length of type 0 input, and each s_i is computable in polynomial time with oracle h).

Therefore, by the characterization theorem of relativized polynomial hierarchy, we have

$[F(h,x)=1] \equiv [\exists |y_{n+1}| \leq |x|^k \forall |y_n| \leq |x|^{k^2} \ldots Q|y_1| \leq |x|^{k^{n+1}} T'(x, y_1, \ldots, y_{n+1}, h)]$

OR

$[F(g,x)=1] \equiv [\exists |y_{n+1}| \leq |x|^k \forall |y_n| \leq |x|^{k^2} \ldots Q|y_1| \leq |x|^{k^{n+1}} T^g_1(x, y_1, \ldots, y_{n+1})]$,

where T^g_1 is a polynomial time predicate with oracle g.

Since $f(x) = F(g,x)$ this implies that $f(x) \in \Sigma^{p,g}_n$, a contradiction. □

In [14] classes **NP**, **Co-NP**, **PH**, **PSPACE** are also defined. We do not give their definition here but make the following observation.

Corollary 1 $\mathbf{NP} \neq \mathbf{Co-NP}$, $\Pi^{\mathbf{P}}_{\mathbf{n}} \cup \Sigma^{\mathbf{P}}_{\mathbf{n}} \subset \Pi^{\mathbf{P}}_{\mathbf{n+1}} \cap \Sigma^{\mathbf{P}}_{\mathbf{n+1}}$, $\mathbf{PH} \subset \mathbf{PSPACE}$.

Proof: There are oracles showing relativized separation among corresponding type 1 classes. Also the definition of these type 2 classes is such that when all the type 1 inputs are fixed to some set A (a $0-1$ valued function) then the resulting classes are the corresponding type 1 classes relativized to oracle A. □

Remark: The proof above does not show separation between $\underline{\Sigma^{\mathbf{P}}_{\mathbf{n+1}}}$ and $\underline{\Sigma^{\mathbf{P}}_{\mathbf{n}}}$. Indeed, the theorem below shows that this hierarchy collapses to $\underline{\mathbf{POLY}}$.

4.2 Collapse of $\underline{\Sigma^p_n}$ hierarchy

Definition 5 *We define k count tape space bounded machines in the same way as the machines to compute functionals of class C_1 defined in [10] except that only their workspace* (not runtime) *needs to be bounded by a polynomial in length of the contents of k^{th} count tape. The space bound also applies to output tape and to oracle query tapes but not to the oracle answer tapes. We define class* **PSPACE′** *to be the class of all type 2 relations whose characteristic functional can be computed by a k count tape $O(m^p)$ space bounded machines for some k and p.*

Let **PSPACE′**$[g]$ be the class **PSPACE′** relativized to type 1 function g and $\underline{\mathbf{PSPACE'}} = \cup_g$ **PSPACE′**$[g]$.

It is easy to see that for all n, $\Sigma^{\mathbf{P}}_{\mathbf{n}}[g] \subseteq$ **PSPACE′**$[g]$ and therefore $\underline{\Sigma^{\mathbf{P}}_{\mathbf{n}}} \subseteq \underline{\mathbf{PSPACE'}}$.

Theorem 2 $\underline{\mathbf{PSPACE'}} = \underline{\mathbf{POLY}}$.

Proof: Let $F(f,n) \in \underline{\mathbf{PSPACE'}}$. There exists a g such that $F(f,n) = G(f,g,n)$ and $G(f_1, f_2, x)$ is computable by a machine k count tape $O(m^p)$ space bounded machine M.

Observe that if the space used by some machine is bounded by Z, then there is an equivalent machine whose runtime can be bounded by c^Z, for some constant c which depends on the machine only. Suppose that L_1 is the length of the largest answer returned by machine M computing G, on input f, g, n. Let $L = max\{|n|, L_1\}$, then space used by M is bounded by L^p, hence time used by it is bounded by c^{L^p}. Consider a function g' which on input of length x returns the output of length c^{x^p}, that is $|g'(n)| = c^{|n|^p}$.

Let g'' be a function such that, $g''(2n) = g(n)$ and $g''(2n+1) = g'(n)$.

Our objective now is to design a *time bounded* count tape machine which on input f, g'', n can simulate M on input f, g, n. This is possible as the function g is coded in the function g''. Further the machine to be designed can also query g'' on appropriate (odd numbered) inputs to receive large enough answer so that its runtime (not just the space used) is also bounded by a polynomial in the length of the largest count tape contents.

Consider a $k+1$ tape $O(m^q)$ machine M', $(q > p, \forall t > q[t^q \geq 2t+1])$, computing $G'(f, h, n)$, which on input f, h, n first queries h on $2.max\{n, p\} + 1$ and starts simulating M on input f, g, n. Whenever M queries g at u, then M' queries h at $2u$ and at $2.h(2u) + 1$ and returns $h(2u)$ to M as an answer. If M queries f at v during the simulation then M' queries f at v and h at $2f(v) + 1$ and returns $f(v)$ as the answer. Further whenever M' queries h at any odd value $2x+1$ then M' halts if $|h(2x+1)| < c^{|x|^p}$.

It is easy to see that on any input f, h, n the *runtime* of M' at all instants during the computation is bounded by a polynomial in $|max\{n, y_1, \ldots, y_i\}|$, where $y_1, \ldots, y_i$ are all the oracle answers returned till that instant. Notice that, since M is a k count tape machine therefore all queries of M' lie in block $[k, k+1]$ or lower.

Therefore by corollary 1, [10] we have that $G'(f, h, n)$ is in **POLY**. Also $F(f, n) = G'(f, g'', n)$. Hence $F(f, n) \in \mathbf{POLY}[g'']$. □

Corollary 2 *For all $n \geq 0$, $\underline{\Sigma_n^P} = \underline{\mathbf{POLY}}$.*

It is of independent interest to note the following.

Observation: A Functional $F(f, x) \in \mathbf{PSPACE}'$ iff it is computable by OTMs in space bounded by $P(|f|, |x|)$, for some second order polynomial P.

This is shown easily because a functional $A(f, x)$, such that $|A(f, x)| = P(|f|, |x|)$, is in $\mathbf{PSPACE}'$.

4.3 Closure under length bounded quantification

It is conjectured in [14] that the levels of the class of polynomially bounded relations are not closed under length bounded existential quantification. We prove a more general result below which implies the conjecture.

Lemma 2 *There is a functional $A(f, z)$, obtained by prefixing $k+1$* **POLY** *bounded existential quantifiers to a* **POLY** *predicate, which is not representable by any POTM computable predicate quantified k times (by any combination of $\exists$ and $\forall$ quantifiers) using POTM functionals bounded quantifiers.*

Proof: Let $A(f, z) = \exists |x_1| \leq |z| \exists |x_2| \leq |f(x_1)| \ldots \exists |x_{k+1}| \leq |f(x_k)| [f(x_{k+1}) \text{ is even }] \ldots (1)$

For each formula

$Q_1 |y_1| \leq |s_1(z, f)| \ldots Q_k |y_k| \leq |s_k(y_1, \ldots, y_{k-1}, z, f)| R(y_1, \ldots, y_k, z, f) \ldots (2)$

where $s_1, \ldots, s_k, R$ are computable by POTMs, and each $Q_i \in \{\exists, \forall\}$, we construct an f such that for some large enough z, formula $A(f, z)$ is true iff (2) is false.

Let t be such that $s_1, \ldots, s_k, R$ are computable by $O(m^t)$ POTMs. We describe the construction of f in the following.

In the first step, we proceed as follows. Choose a z such that $|z|^t < 2^{|z|}$. Run a POTM computing s_1 on input f, z answering all queries to f as 1. Define f to be 1 on all the inputs queried during the computation. Define f to be an odd number M_1 at all unqueried strings of length $\leq |z|$. M_1 is chosen such that it satisfies the property $2^{|M_1|} > |M_1|^t \times 2^{|z|^t}$. Let v_1 be one string such that $|v_1| \leq |z|$ and $f(v_1) = M_1$. (v_1 exists by the choice of z).

In the second step we proceed as follows. For all $y_1, |y_1| \leq |z|^t$ run a POTM computing $s_2(y_1, f, z)$ and answer all queries to f at inputs where f has not been defined so far, as 1. f is defined to be 1 at the queried string, after answering such a query. After the computation of s_2 define f at all the strings of length $\leq |M_1|$, on which f is not defined, as M_2. M_2 is chosen to be an odd number such that $2^{|M_2|} > |M_2|^t \times 2^{|M_1|^t}$. Let v_2 be a string such that $|v_2| \leq |M_1|$, and $f(v_2) = M_2$, such a v_2 exists by choice of M_1 at the previous step.

In the i^{th} step we do the following; for all sequences $y_1, y_2, \ldots, y_{i-1}, z$ such that $|y_1| \leq |M_{i-2}|^t, \ldots \ldots, |y_{i-1}| \leq |M_{i-2}|^t$, run a POTM computing $s_i(y_1, \ldots, y_{i-1}, z, f)$. During this computation if f is queried at an input on which it has not been defined then 1 is returned as the answer to the query and f is defined at the queried string to be 1. At the end of this simulation, we choose an odd number M_i for which $2^{|M_i|} > k^2 \times |M_i|^t \times 2^{|M_{i-1}|^t}$ holds. Define f to be at all the inputs of length $\leq |M_{i-1}|$, where it has not been defined so far, as M_i. Let v_i be one string such that, $|v_i| \leq |M_{i-1}|$ and $f(v_i) = M_i$. (by the choice of M_{i-1} at the previous step such a string exists, as total number of strings queried in the simulation upto and including i^{th} step is $\leq k^2 \times |M_{i-1}|^t \times 2^{|M_{i-2}|^t}$)

After k steps we obtain M_k.

Notice that by the preceding construction, the truth of (2) on input z, f depends only on $R(y_1, \ldots, y_k, z, f)$ such that $|y_1| \leq |M_{k-1}|^t, \ldots, |y_k| \leq |M_{k-1}|^t$. Determine the truth of formula (2) by running a POTM computing $R(y_1, \ldots, y_k, z, f)$ on all $y_i's$ such that $|y_i| \leq |M_{k-1}|^t$ and answering all queries to f where it is not defined as 1. f is defined to be 1 at such a queried input after answering the query. Observe that M_k is chosen to be large enough so that there is a string of length $\leq |M_k|$ which is not queried by R on any input and is also not queried in any of the previous k steps and f is not defined on this string.

Let u be a string of length $\leq |M_k|$, not queried in any of the previous steps and on which f is not defined. Define $f(u)$ as 4 (some even value) iff the formula (2) is false. Define the value of f to be 1 at all the inputs on which it has not been defined so far. (By choice of u, truth of (2) is independent of $f(u)$).

If the formula (2) is true then $A(f, z)$ is false as $f(x)$ is odd for all x, on the other hand if the formula (2) is false then (1) is true via $x_1 = v_1, x_2 = v_2, \ldots x_k = v_k, x_{k+1} = u$. □

Corollary 3 *Levels of the hierarchy of polynomially bounded relations are not closed under* POLY *length bounded existential quantification of natural number variables. In fact, for each level, there is an infinite hierarchy based on the number of* POLY *bounded existential quantifiers that can prefix formulas of that level.*

5 Feasible reducibility between functionals

In order to define the polynomial hierarchy for type 2 functionals we need to define some notion of resource bounded reduction between functionals. Below we try to generalize the notion of poly-time many one and poly-time Turing reduction for type 1 functions to the type 2 case.

Definition 6 *A functional $F(f, x)$ many one reduces to a functional $G(f, x)$, if there are basic feasible functionals ϕ_1, ϕ_2, ϕ_3 such that $F(f, x) = \phi_1(G(\lambda n \phi_2(f, x, n), \phi_3(f, x)), f, x)$. The generalization of this notion to the case where F and/or G have more than one function argument etc., is obvious.*

Definition 7 *A functional F Turing reduces to a functional (or function) G, if F can be represented by a λ term constructed from G using basic feasible functionals and λ calculus operations.*

A machine characterization of this reducibility can be obtained by machines described in [12] to compute type 3 functionals. Specifically, F Turing reduces to G iff F can be computed by a k count tape HPOTM which uses only finitely many parameterizable functionals for querying and whose all type 2 inputs have been fixed to G.

We shall use a weaker notion than Turing reduction to define a version of polynomial hierarchy. The idea of this weaker Turing reduction is that if F reduces to G than F can be computed at input f, x using G as oracle such that whenever G is queried during this computation its type 1 input is a basic feasible function in f. This reduction is weaker than Turing reduction as it does not allow G to be queried at type 1 input which is a basic feasible function in G, f.

Definition 8 *F weakly Turing reduces to G iff there is a k count tape HPOTM M, designed to compute a type 3 functional, such that it uses finitely many parameterizable functionals,* none of which takes type 2 inputs of M as parameters, *for querying. Further M computes F when all of its type 2 inputs have been fixed to G.*

It is correct to say that reductions defined above are reducibilities as they are reflexive and transitive. All the reducibilities defined above differ on the space of computable type 2 functionals. Turing reducibility is the most general and many-one reducibility is the least general. A many-one reduction is a very special case of Turing reduction.

Lemma 3 *There are computable functionals F, G such that F weakly Turing reduces to G but F does not many one reduce to G.*

Proof: This is plausible if we notice that a many one reduction can query G only at one input but a weak Turing reduction can query G at a several inputs before outputting the value of F on a given input.

Let G be a type 2 functional of type $[N \to N] \to N \to N$.

Define function f as follows:

$f(x) = 1$ if both $G(\lambda x.x, x)$ and $G(\lambda x.x + 1, x)$ are odd or both are even.

$= 0$ otherwise

f weakly Turing reduces to G, for every G. However a computable G can be designed easily by diagonalizing over all triples of basic feasible functionals such that f does not many one reduce to G.

G is designed as follows.

Inputs to G are g, n.

For all g, n such that $g(1) \notin \{1,2\}$, $G(g,n)$ is defined to be 1.

For other cases G is constructed in stages, each stage corresponding to a natural number.

(with i^{th} stage there is a constant n_i associated, $n_1 = 1$)

i^{th} stage:

Let i^{th} triple of BFF be $< \phi_1, \phi_2, \phi_3 >$.

1. **Case $G(\lambda n.\phi_2(n,n_i), \phi_3(n_i))$ is defined :**
   if $\phi_1(G(\lambda n.\phi_2(n,n_i), \phi_3(n_i)), n_i) = 0$
   then define $G(g,n_i) = 1$ whenever $g(1) \in \{1,2\}$
   else define $G(g,n_i) = g(1)$ whenever $g(1) \in \{1,2\}$.
2. **Case $\phi_2(1,n_i) = 1$:**
   Define $G(\lambda n.\phi_2(n,n_i), \phi_3(n_i)) = 1$ and evaluate $\phi_1(1,n_i)$
   if $\phi_1(1,n_i) = 0$
   then define $G(g,n_i) = 1$ whenever $g(1) \in \{1,2\}$
   else define $G(g,n_i) = g(1)$ whenever $g(1) \in \{1,2\}$.
   [Notice that these assignments to G are consistent with the earlier assignments to it.]
3. **Case $\phi_2(1,n_i) = 2$:**
   Define $G(\lambda n.\phi_2(n,n_i), \phi_3(n_i)) = 1$ and evaluate $\phi_1(1,n_i)$
   if $\phi_1(1,n_i) = 0$
   then define $G(g,n_i) = 1$ whenever $g(1) \in \{1,2\}$
   else define $G(g,n_i) = g(1) - 1$ whenever $g(1) \in \{1,2\}$.

Let $n_{i+1} = max\{n_i, \phi_3(n_i)\} + 1$.

Define $G(g,x) = 1$ if $g(1) \in \{1,2\}$ and $x < n_{i+1}$ and $G(g,x)$ is not defined so far.

End of stage i .

It is not difficult to see that the above construction yields a well defined and total computable functional G. In the i^{th} stage we made sure that f does not many one reduce to G via the i^{th} triple of the BFFs . Therefore f does not many one reduce to G. □

Turing reduction is more general than the weak Turing reduction will be shown in section 7.

For generating the polynomial hierarchy, we need to iterate using a nondeterministic operator. Therefore we define a nondeterministic version of the weak Turing reduction.

Definition 9 *F, a $0-1$ valued functional, weakly nondeterministic Turing reduces to G iff there is a* nondeterministic *k count tape HPOTM M, designed to compute a type 3 functional such that it uses finitely many parameterizable functionals,* none of which takes type 2 inputs of M as parameters, *for querying. Further M computes F if all its type 2 inputs have been fixed to G.*

In section 7 we define a polynomial hierarchy using weak nondeterministic Turing reduction. We do not define a nondeterministic version of Turing reduction as it generates a hierarchy which collapses at the first level. This will be shown in section 7.

6 Victor Harnik's type 2 polynomial hierarchy

In [9], Victor Harnik used a notion of polynomial hierarchy for all finite types in order to extend the results of [5]. In this section we study the type 2 section this hierarchy in some detail. Buss has also defined a similar hierarchy in [3], although using Gödel numbering.

Definition 10 *[9] $\Box^p_{2,i}$= All functionals in the λ closure of BFF and Σ^p_i, where Σ^p_i is the i^{th} level of the type 1 polynomial hierarchy.*

This hierarchy does not specify the $\Sigma^p_{2,i}, \Pi^p_{2,i}$ levels explicitly. For the sake of completeness we can take $\Sigma^p_{2,i} = \Sigma^p_i$ and $\Pi^p_{2,i} = \Pi^p_i$.

The most important property of this hierarchy, which is required in proving the results of [9], is that it preserves $\Box^p_i$. That is, all type 1 functions constructed using functionals in $\Box^p_{2,i}$, functions in $\Box^p_i$ and the λ calculus operations are in $\Box^p_i$. In fact a slightly more general property can be proved:

Theorem 3 *Let C be a complexity class of type 1 functions which contains $\Box^p_i$ and is closed under composition. If C can be expressed as $C = \cup_i DTIME(t_i(n))$, where t_i's are monotone and time constructible functions then C is preserved by $\Box^p_{2,i}$.*

Proof: Similar to that of theorem 3, [11]. □

Observation: The above type 2 hierarchy collapses iff the type 1 polynomial hierarchy collapses.

This is because if type 1 poly hierarchy collapses, say $\Box^p_i = \Box^p_{i+1}$ then by definition $\Box^p_{2,i} = \Box^p_{2,i+1}$ hence the given type 2 hierarchy also collapses. On the other hand, if $f \in \Box^p_{i+1} - \Box^p_i$ then $f \in \Box^p_{2,i+1} - \Box^p_{2,i}$ also, as all type 1 functions in $\Box^p_{2,i}$ are in $\Box^p_i$.

It is possible to give a machine characterization of this class using count tape machines of [10] which define C_1 or using the characterization of basic feasible functionals by Cook and Kapron in [6].

Lemma 4 *A functional $F \in \Box^p_{2,i+1}$ iff it can be computed by some count tape machine M of class C_1 as in [10] where some type 1 inputs of M may be fixed by $\Box^p_{i+1}$ functions.*

Proof: The proof is similar to the proof of lemma 6, [11] and follows from the definition of $\Box^p_{2,i+1}$. □

Since the output size of $\Box^p_i$ is bounded by a polynomial it is possible to characterize $\Box^p_{2,i}$ using second order polynomials. We state this below without proof.

Lemma 5 *A functional is in $\Box^p_{2,i}$ iff it can be computed by a type 2 oracle TM using a Σ^p_i complete set as oracle in time bounded by a second order polynomial in the length of type 0 and type 1 inputs to the functional.*

Each $\Box^p_{2,i}$ a has complete problem, namely Σ^p_i complete problem (which can also be thought of as a type 2 functional) under both Turing and the weak Turing reduction. The two reductions coincide as we are reducing to a type 1 function. Do $\Box^p_{2,i}$ have many one complete problems? A negative answer implies $P \neq NP$.

We can define another type 2 poly-hierarchy whose i^{th} level is obtained by taking the λ closure of functionals of class C_2 with $\Box^p_i$, this hierarchy also preserves the $\Box^p_i$ functions. This leads to a somewhat informal, general conjecture.

Conjecture: Let C be the largest class of feasible functionals with all "desirable preservation properties" then the type 2 poly-hierarchy obtained by taking the λ closure of functionals of C with the corresponding level of type 1 poly hierarchy functions yields the **largest** hierarchy with all the "desirable preservation properties".

7 Type 2 polynomial hierarchy PH'

In this section we define another version of polynomial hierarchy for type 2 relations. Unlike the hierarchy of the previous section levels of this hierarchy are closed under appropriate second order polynomial bounded quantification. This hierarchy has two natural characterizations, machine based and quantifier based.

We first introduce a simple machine model to define this hierarchy, then present its alternative characterizations and study some of its properties-such as existence of complete sets for various levels.

A nondeterministic polynomial time oracle Turing machine (NPOTM) is a nondeterministic oracle Turing machine such that the computation time on each path is bounded by a fixed polynomial in the length of the maximum of the largest type 0 input and the largest answer returned on that path. Further, computation on each path is required to terminate.

For defining our version of polynomial time hierarchy we need to introduce relativized oracle Turing machines (ROTM). An ROTM computes a functional relative to some functional of same arity (that is the functional with the same number of type 0 inputs the same number of type 1 inputs). Functional oracle is accessed on a special functional oracle tape by writing type 0 input to the functional on this tape. Function inputs for both functionals, the functional being computed and the functional being queried, are taken to be the same. To clarify the above concept we give an example. An OTM computing $F(f, n)$ relative to functional $G(g, m)$ has an oracle tape through which function f is accessed. Apart from this the machine also has a functional oracle tape through which G is accessed by writing type 0 input for functional G. If OTM computing F is running on input f, n and queries G by writing x on functional oracle tape during the computation then the value of $G(f, x)$ appears on the functional oracle answer tape. In our case all functional oracles will be relations. RPOTM and RNPOTM are defined in an obvious manner. Let NP' be class of all functionals computable by NPOTMs such that the runtime on each path is bounded by a second order polynomial.

We can now define polynomial hierarchy for type 2 in analogy with polynomial hierarchy for type 1 as follows. We denote the levels of this hierarchy with the usual symbols for polynomial hierarchy, but with superscript $'$.

P' = basic feasible functionals = Σ'_0=Π'_0

$\Sigma'_{n+1} = NP^{\Sigma'_n}$ = functionals computable by RNPOTMs in second order polynomial time with relations from Σ'_n as oracle.

$\Pi'_{n+1} = Co - \Sigma'_n$

$\Delta'_{n+1} = P^{\Sigma'_n}$ = functionals computable by RPOTMs in second order polynomial time with relations from Σ'_n as oracle.

$PH' = \cup_{n\geq 1} \Sigma'_n$

By the argument of lemma 1 all the Σ'_n, Π'_n classes are different. This argument is applicable because when all type 1 inputs are fixed to a 0-1 valued set A, then the classes in the above hierarchy are the corresponding type 1 complexity classes relativized with oracle A. Oracles that separate type 1 polynomial hierarchy are known to exist.

Σ'_n (Π'_n) levels of this version of polynomial hierarchy are easily seen to be closed under second order polynomial bounded existential (universal) quantification.

Lemma 6 *A functional is in Σ'_k iff it can be expressed by quantifying a basic feasible functional by second order polynomial bounded quantifiers with at most k alternations of quantifiers such that the formula begins with an existential quantifier. A functional is in Π'_k iff it can be expressed by a formula as above except that it must start with a universal quantifier now.*

Proof: Observe that for each second order polynomial $P(f,x)$ there is a type-2 NP machine which works in second order polynomial time and on input f,x computes $P(|f|,|x|)$ on some branch and on all other branches a number $\leq P(|f|,|x|)$.

The rest of the proof is a direct generalization of the characterization theorem for polynomial hierarchy of type 1. (see proofs of theorems 8.2 & 8.3 in [1]). □

There is also an alternative way to present Δ' levels of the hierarchy PH' as shown in the next lemma. For terms not defined here, see [14].

Lemma 7 *For $n \geq 0$, Δ'_{n+1} is the class of functionals obtained from functionals of class Σ'_n by application, expansion, functional composition and limited recursion on notation.*

Note that functional abstraction is not included in the above lemma.

Proof (sketch): Let $F(f,n) \in \Delta'_{n+1}$ and let M be a RPOTM computing $F(f,n)$ with oracle $G(f,n)$, where $G(f,n) \in \Sigma'_n$. From M it is easy to design a POTM M' such that M' computes a functional $F'(f,g,n)$ in time $P(|f|,|g|,|n|)$ for some second order polynomial P. Further $F(f,n) = F'(f, \lambda x G(f,x), n)$. By [7], F' is a basic feasible functional. By using the same reasoning as in section 4, [4] it can be constructed from some initial functions using application, expansion, functional composition and limited recursion on notation. By induction on the length of such derivation of F', $F'(f, \lambda x G(f,x), n)$ can be shown to be constructible by application, expansion, functional composition and limited recursion on notation from the functionals in Σ'_n. □.

We now turn our attention to the existence of complete sets for the levels of hierarchy PH'.

Lemma 8 *Let C be a class of type 2 relations, such that C is contained in class of relations obtained from POTM predicates using POTM functional bounded existential quantification. If C contains* POLY *and is closed under* POLY *bounded existential quantification then C has no complete relation under many one reduction.*

Proof: Let $R(f,x) \in C$ be a complete relation for C. Then R is k quantifier relation for some k. (that is, $R(f,x)$ can be represented as a formula obtained from a POTM computable predicate by prefixing it with k POTM computable bounded quantifiers)

We use the similar argument as in lemma 2 to derive a contradiction.

We can demonstrate a $k+2$ existential quantifier formula $A(f,x)$ which does not many one reduce to R by any triple ϕ_1, ϕ_2, ϕ_3 of basic feasible functionals.

For each candidate triple ϕ_1, ϕ_2, ϕ_3 of basic feasible functionals we construct a function f such that formula $\phi_1(R(\lambda n \phi_2(f,x,n), \phi_3(f,x)), f, x)$ is true iff the formula A is false.

The construction of f is very similar to the construction in lemma 2, so we omit the details here. □

Corollary 4 *For all $k \geq 1$, Σ'_k does not have any complete set under many one reduction.*

Proof : Using same idea as in the lemma above. □

Remark: We do not know if Σ'_k have complete sets under feasible Turing reduction.

Although our definition of RPOTMs and RNPOTMs is very simple, the above hierarchy is same as the one generated using weak nondeterministic Turing operator.

Theorem 4 *Σ'_{n+1}=set of predicates which are weak nondeterministic Turing reducible to Σ'_n.*

Proof: That Σ'_{n+1} contains only weak nondeterministic reducible predicates to Σ'_n is obvious by the definition of Σ'_{n+1} as RNPOTMs are specialized machines which witness weak nondeterministic Turing reduction to type 2 functionals. To prove the other direction we first show that if $F(f,x) \in \Sigma'_n$ and $\phi_1(f,x,y), \phi_2(f,x)$ are BFF then $H(f,x,u) = F(\lambda y \phi_1(f,x,y), \phi_2(f,u)) \in \Sigma'_n$. This can be proved by induction on n. For the base case let $F(f,x) \in \Sigma'_1$, the machine to compute $H(f,x,u)$ in this case is designed by modifying the machine M which computes $F(f,x)$, for each query of the machine M to f at y we answer the query by running $\phi_1(f,x,y)$. For the induction step let $F(f,x) \in \Sigma'_{n+1}$ be computed by RNPOTM M' with an oracle $G(f,z) \in \Sigma'_n$. Let $G'(f,x,z) = G(\lambda y \phi_1(f,x,y), z)$, by induction hypothesis $G' \in \Sigma'_n$. Consider a RNPOTM machine M'' with oracle G' which on input f,x,u starts simulating M' with type 0 input as $\phi_2(f,u)$, M'' answers query of M' to type 1 input at argument m as $\phi_1(f,x,m)$. A query of M' to type 2 oracle at argument m is converted by M'' to a query at argument

x, m to its type 2 oracle. It is easy to see that M'' with its type 2 oracle as G' computes $H(f, x, u)$ and can be made to work in time polynomial in $|f|, |x|, |u|$. Hence $H \in \Sigma'_{n+1}$.

Now let some machine M_1 weakly nondeterministic Turing reduce a set A to functional $F(f, x) \in \Sigma'_n$. Since M_1 uses only finitely many query functionals, M_1 can be thought of querying functionals $F'_1, \ldots, F'_i \in \Sigma'_n$, obtained by the construction of the preceding paragraph, at function input f only. Therefore M_1 can be modified to an RNPOTM computing set A relative to functionals $F'_1, \ldots, F'_i$ in second order polynomial time. □

Theorem 5 $\cup_{n\geq 0}\Sigma'_{n+1} \subsetneq BFF(\Sigma'_1)$

Proof: The idea is similar to that in the proof of lemma 1. We just give an example. We show that $A(f, z) = \exists x \leq \phi_1(f, z)\forall y \leq \phi_2(f, z, x)[G(f, x, y, z)]$, where ϕ_1, ϕ_2, G are BFF is always in BFF(Σ'_1).

Let functional $F(f, g, z) = \exists x \leq \phi_1(f, z)[g(z, x) \neq 1]$, $F \in \Sigma'_1$.

Let $\underline{H}(f, z, x)$ be the characteristic function of predicate $\exists y \leq \phi_2(f, z, x)[\neg G(f, x, y, z)]$.

$\underline{H}(f, z, x) \in \Sigma'_1$.

Now $F(f, \lambda z\lambda x\underline{H}(f, z, x), z) = \exists x \leq \phi_1(f, z)\forall y \leq \phi_2(f, z, x)[G(f, x, y, z)] =A(f, z)$.

Therefore $A(f, z) \in BFF(\Sigma'_1)$.

By repeating this argument it is easily shown that $\cup_{n\geq 0}\Sigma'_{n+1} \subsetneq BFF(\Sigma'_1)$. □

Corollary 5 *Turing reduction is more powerful than weak Turing reduction.*

Proof: Since $\Sigma'_3 \neq \Sigma'_2$, by theorem 4 it follows that there is a Functional in Σ'_3 which does not weak Turing reduce (in fact it does not even weak *nondeterministic* Turing reduce) to Σ'_1. However by previous theorem every functional in Σ'_3 Turing reduces to Σ'_1. □

8 PH' equals polynomially bounded relations

Though each level of PH' is larger than the corresponding level of the class of polynomially bounded relations of section 3, the following theorem shows that both contain the same relations.

Theorem 6 *PH' is contained in the class of polynomially bounded relations.*

Proof: Let R be a relation in PH'. R can be represented as a basic feasible functional prefixed by a string of second order polynomial bounded quantifiers. We can obtain a relation equivalent to R by replacing each second order quantifier by a string of **POLY** bounded quantifiers, as indicated below.

Observe that any second order polynomial can be simulated by a string of **POLY** length bounded quantifiers. To illustrate by an example let the second order polynomial be $P(f, x) = f(f(x^j)+x^j)+x^j$.

$\exists|z| \leq P(|f|, |x|)$ is equivalent to $\exists|x_1| \leq |x|^j\exists|x_2| \leq |f(x_1)| + |x|^j\exists|z| \leq |f(x_2)| + |x|^j$.

in the sense that the former can be replaced by the latter in any formula (in which the variables x_1, x_2 do not appear) without changing the meaning of the formula.

In the same manner $\forall|z| \leq P(|f|, |x|)$ is equivalent to

$\forall|x_1| \leq |x|^j\forall|x_2| \leq |f(x_1)| + |x|^j\forall|z| \leq |f(x_2)| + |x|^j$.

After replacing each quantifier by the process indicated above we obtain a relation equivalent to R which involves only **POLY** bounded quantifiers. R is therefore in the class of polynomially bounded relations.

General construction of **POLY** bounded quantifier strings for simulating second order polynomials (with length of type 0 and type 1 objects as input to the polynomial) can be defined inductively along with the definition of second order polynomials. □

9 Conclusion

In this paper we have shown limitations on finding a faithful type 2 analog of type 1 poly-hierarchy. We then studied two polynomial hierarchies which seem natural from different viewpoints. The hierarchy of section 6 is properly contained in the hierarchy PH', so perhaps Victor Harnick's hierarchy may be called "basic polynomial hierarchy". However we need to figure out what the key criteria for defining type 2 poly-hierarchy should be to make some progress. The subject of feasible reduction between type 2 functionals also requires further study.

We also solved most of problems posed in the work of Townsend [14]. Townsend [14] suggested the use of topological concepts for solving these problems but we solve them without involving such concepts. In fact, we can obtain some results about open sets introduced in [14] from our work. For instance $A \in \underline{\mathbf{POLY}}$ iff A is $(\underline{\mathbf{POLY}}, \underline{\mathbf{POLY}}, \underline{\mathbf{POLY}})$ open iff A is $(\underline{\mathbf{POLY}}, \underline{\mathbf{POLY}}, \underline{\mathbf{POLY}})$ closed. However topological concepts may be useful in studying fine structure and interrelationships among type 2 complexity classes. It should be possible to examine a hierarchy using the function quantifiers. This can give a feasible analog of analytical hierarchy, which perhaps can be used in effective descriptive set theory.

References

1. Balcazar, Diaz, and Gabarro. *Structural Complexity I.* Springer-Verlag, 1988.
2. S. Buss. *Bounded Arithmetic.* Bibliopolis, Naples, 1986.
3. S. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In *Structure in Complexity Theory*, pages 77–103. Springer-Verlag, Lecture Notes in Computer Science No. 223, 1986.
4. P. Clote, A. Ignjatovic, and B. Kapron. Parallel computable higher type functionals. In *Full version*, July 1994.
5. S. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, pages 103–200, Volume 63, 1993. *Extended abstract in STOC89.*
6. S. A. Cook and B. M. Kapron. Characterizations of the basic feasible functionals of finite type. In *Proceedings of MSI Workshop on Feasible Mathematics, S. Buss and P. J. Scott, editors, perespective in computer science, Birkhauser-Boston, New York*, pages 71–95, 1990.
7. S. A. Cook and B. M. Kapron. A new characterization of Mehlhorn's polynomial time functionals. In *FOCS*, 1991.
8. Stephen A. Cook. Computability and complexity of higher type functions. In *MSRI Proceedings*, 1990.
9. Victor Harnik. Provably total functions of intuitionistic bounded arithmetic. *Journal of Symbolic Logic*, pages 466–477, 1992.
10. A. Seth. There is no recursive axiomatization for feasible functionals of type 2. In *Seventh Annual IEEE Symposium on Logic in Computer Science*, 1992.
11. A. Seth. Some desirable conditions for feasible functionals of type 2. In *Eighth Annual IEEE Symposium on Logic in Computer Science*, 1993.
12. A. Seth. Turing machine characterizations of feasible functionals of all finite types. In *Proceedings of MSI Workshop on Feasible Mathematics, P. Clote and J. Remmel, editors, perespective in computer science, Birkhauser-Boston, New York*, 1994.
13. L. J. Stockmeyer. The polynomial time hierarchy. *Theoretical Computer Science*, pages 1–22, 1976.
14. M. Townsend. Complexity for type-2 relations. *Notre Dame Journal of Formal Logic*, pages 241–262, 1990.
15. A. Yao. Separating the polynomial-time hierarchy by oracles. In *IEEE Symposium on Fondations of Computer Science*, 1985.

The Hierarchy of Terminating Recursive Programs over N

Stanley S. Wainer *

Pure Mathematics Dept., Leeds University, Leeds LS2 9JT, UK

1 Introduction

A terminating recursive program defines a total recursive functional, taking "given" functions to the function defined from them by the program. Termination means that the program has a well-founded computation tree with a recursive ordinal as its height, and Kleene (1958) noted that, in contrast with the well known "collapsing phenomenon" for hierarchies of recursive functions, the resulting hierarchy expands right the way through ω_1^{CK} : i.e. for each recursive ordinal α there is a total recursive functional which cannot be defined by any program of height less than α.

In this paper we examine some of the ways in which the ordinal height of a program encodes its complexity. By a careful assignment of (proof theoretic) ordinal bounds to derivations in Kleene's equation calculus, the standard "fast", "medium" and "slow" growing hierarchies emerge as canonical complexity measures allowing different forms of recursion to be classified and compared. Known relationships between these hierarchies then yield measures of "transformational complexity" (e.g. recursive to tail recursive) in terms of their corresponding ordinal trade-offs. The underlying theme is that of Cut Elimination, but in an equational setting.

2 Kleene's Hierarchy of Recursive Functionals

A recursive program, defining a (possibly partial) function $f : N^k \to N$ from "given" functions $g_1, \ldots, g_m$, can be viewed as a monotone, continuous operator Φ on partial functions, having f as its least fixed point

$$f \;=\; \Phi(g_1, \ldots, g_m, f)\ .$$

*Partially supported by an EU Science Plan Twinning Project between Leeds, Munich and Oslo, entitled "Proof Theory and Computation". The author thanks D. Leivant and H. Schwichtenberg for valuable comments on an earlier draft.

To simplify notation, we shall regard the given functions as having been coded into a single unary function $g(z) = \langle g_1(z), \ldots, g_m(z)\rangle$, and similarly the arguments of f as having been coded into a single argument $x = \langle x_1, \ldots, x_k\rangle$. Thus the fixed-point equation defining f is just

$$f(x) = \Phi(g, f)(x) .$$

The solution to this equation is

$$f = \bigcup_i f_i$$

where $f_i = \Phi(g)^i(\perp) = \Phi(g, \ldots \Phi(g, \Phi(g, \perp)) \ldots)$ so that $f_i \subseteq f_{i+1}$ for every $i = 0, 1, 2, \ldots$ Therefore for all $n, m \in N$,

$$f(n) = m \quad \Leftrightarrow \quad \exists i \; (f_i(n) = m)$$

and since the successful evaluation of $f_i(n) = \Phi(g)^i(\perp)(n)$ will require only finitely many calls on g, say $g(0), g(1), \ldots, g(j-1)$, we can go a step further and write

$$f(n) = m \quad \Leftrightarrow \quad \exists i \, \exists j \; (\Phi(\bar{g}(j))^i(\perp)(n) = m)$$

where $\bar{g}(j)$ denotes the finite subfunction of g from $g(0)$ up to $g(j-1)$. Furthermore since Φ is monotone we can contract the two existential quantifiers into one by taking any $y \geq \max(i, j)$ thus

$$f(n) = m \quad \Leftrightarrow \quad \exists y \; (\Phi(\bar{g}(y))^y(\perp)(n) = m) .$$

¿From this we can immediately derive Kleene's fundamental Normal Form for recursion. Below we let σ, τ range over the set of all finite sequences $\langle x_0, \ldots, x_{j-1}\rangle$ of natural numbers, so that $\sigma = \bar{g}(j)$ means $g(i) = x_i$ for each $i < j$. We let $lh\sigma$ denote the length of σ, $\sigma \frown z$ denote the sequence obtained by appending z to the end of σ, and $\sigma \frown \tau$ denote the new sequence obtained by concatenating the two etc.

Theorem 2.1 *Let U be any recursive surjection from N to N such that $U^{-1}(n)$ is infinite for every n. Then with each operator Φ as above, we can associate a recursive relation*

$$T_\Phi \; (n, \sigma, y) \;\; :\equiv \;\; \Phi(\sigma)^y(\perp)(n) \downarrow = U(y)$$

such that if f is recursively defined from a given function g as the least fixed point of the equation

$$f = \Phi(g, f)$$

then f is expressible in the normal form

$$f(n) = U(\, \mathit{least} \; y . \, T_\Phi(n, \bar{g}(y), y) \,) .$$

Remark. T_Φ would normally be *primitive* recursive. Furthermore if we parameterized T by introducing a new variable e ranging over codes for operators Φ, then it would become a version of Kleene's "T-predicate".

Definitions. With each operator Φ as above, associate the following *partial recursive functional* F_Φ from partial functions g to partial functions f,

$$F_\Phi(g) \ := \ \text{the least fixed point } f \text{ of the equation } f = \Phi(g, f)\,.$$

Call Φ a *terminating operator* and F_Φ a *total recursive functional* if for each totally defined function $g : N \to N$ the output function $f = F_\Phi(g)$ is also totally defined.

Theorem 2.2 *The normal form for F_Φ is*

$$F_\Phi(g)(n) \ = \ U(\ \textit{least}\ y.\, T_\Phi(n, \bar{g}(y), y)\ )\,.$$

and therefore

$$F_\Phi \textit{ is a total recursive functional } \ \Leftrightarrow \ \forall g\, \forall n\, \exists y\, T_\Phi(n, \bar{g}(y), y)\,.$$

Definition. With each operator Φ as above, associate the following tree of finite sequences of natural numbers :

$$\mathrm{Tree}(\Phi) \ := \ \{\, \langle x_0, \ldots, x_k \rangle \ : \ \forall y < k\ \neg T_\Phi(x_0, \langle x_1, \ldots, x_y \rangle, y)\,\}\,.$$

Tree(Φ) is called the *computation-tree* of Φ.

We imagine Tree(Φ) as growing downwards by extension, i.e. if σ and τ are any two nodes on the tree then σ comes below τ iff σ is an extension of τ. An infinite branch of the tree is thus determined by a number n and a total function $g : N \to N$ such that $\forall y\ \neg T_\Phi(n, \bar{g}(y), y)$. Therefore an infinite branch on Tree(Φ) is a witness to the fact that for some n and g, $F_\Phi(g)(n)$ is not defined. To say that the tree is "well-founded" is to say that there are no infinite branches, and hence :

Theorem 2.3

$$\Phi \textit{ is terminating } \ \Leftrightarrow \ \textit{Tree}(\Phi) \textit{ is well-founded.}$$

This equivalence is the basis for a natural theory of ordinal assignments. It allows us to assign to each terminating operator Φ a (recursive) ordinal $\|\Phi\|$ measuring the size of its computation-tree.

Definition. For each node σ on Tree(Φ) define

$$\begin{array}{lcll} \|\sigma\| & = & 0 & \text{if } \sigma \text{ is a leaf} \\ & = & \sup\ \{\ \|\tau\| + 1 : \tau \text{ extends } \sigma\ \} & \text{otherwise.} \end{array}$$

Then define $\|\Phi\| := \|\langle\rangle\|$ where $\langle\rangle$ is the empty sequence.

Example. The iteration-from-0 operator

$$\Phi(g,f)(n) := \textbf{if } n=0 \textbf{ then } 0 \textbf{ else } g(f(n-1))$$

computes the total recursive functional

$$F_\Phi(g)(n) = g^n(0) .$$

The terminal node or "leaf" on the branch through Tree(Φ) determined by n and g will be the shortest sequence of the form $\langle n, g(0), \ldots, g(y-1)\rangle$ such that $g^i(0) < y$ for every $i \le n$, and $U(y) = g^n(0)$. In particular, if g is such that $g^i(0) < g^{i+1}(0)$ for every $i < n$ then this sequence will be of the form

$$\langle n, g(0), \ldots, g^2(0), \ldots, g^3(0), \ldots, g^{n-1}(0), \ldots, g^n(0), \ldots, g(y-1)\rangle$$

where the value of $g^i(0)$, once fixed, determines the length of the ensuing segment up to $g^{i+1}(0)$. Therefore as we take the supremum over all branches issuing from a given node

$$\langle n, g(0), \ldots, g(g^{i-1}(0)-1)\rangle$$

the successive segments $g^i(0), \ldots, g^{i+1}(0)$ have unbounded length, depending on the value of $g^i(0)$. So each such segment adds one more ω to the ordinal height of the tree. Since there are n such segments, the height of the subtree below node $\langle n\rangle$ must be $\omega \cdot n$. Therefore the height of the entire tree is

$$\|\Phi\| = \sup_n \omega \cdot n = \omega^2 .$$

Definition. For each recursive ordinal α define

$$REC_2(\alpha) = \{ F_\Phi : \Phi \text{ terminating and } \|\Phi\| < \alpha\}.$$

Kleene (1958) noted that, in sharp contrast with any inductive hierarchy for all recursive functions, this hierarchy of total recursive functionals does not depend on any system of ordinal notations and furthermore it does not collapse.

Theorem 2.4 *For each recursive ordinal α there is a total recursive functional which does not belong to $REC_2(\alpha)$.*

Proof. Assume α is a fixed recursive ordinal and choose a recursive well-ordering $\prec_\alpha$ with that order type. Choose also a Gödel numbering of all partial recursive operators $\Phi \mapsto e$. Define a recursive functional $V_\alpha(e,f,g,n,\sigma)$ as follows,

$$\begin{array}{rcll} V_\alpha(e,f,g,n,\sigma) & = & \textbf{if} & \neg T_\Phi(n,\sigma,lh\sigma) \text{ and} \\ & & & f(\langle n\rangle \frown \sigma \frown g(lh\sigma)) \prec_\alpha f(\langle n\rangle \frown \sigma) \\ & & \textbf{then} & V_\alpha(e,f,g,n,\sigma \frown g(lh\sigma)) \\ & & \textbf{else} & U(lh\sigma) . \end{array}$$

Notice that the well-foundedness of $\prec_\alpha$ ensures that V_α is total. Furthermore if $\|\Phi\| < \alpha$ then there will be an order-preserving map $f : Tree(\Phi) \to \alpha$ so that for all g and n,

$$V_\alpha(e, f, g, n, \langle\rangle) \;=\; F_\Phi(g)(n) \;.$$

Consequently the total recursive functional F defined by

$$F(g)(n) \;=\; V_\alpha(g(0), \lambda x. g(x+1), g, n, \langle\rangle) + 1$$

cannot lie in $REC_2(\alpha)$. For if it did there would be an operator Φ with Gödel number e defining it, and an order-preserving map $f : Tree(\Phi) \to \alpha$, so that for the fixed function argument g with $g(0) = e$ and $g(x+1) = f(x)$ we'd have

$$F(g)(n) \;=\; V_\alpha(e, f, g, n, \langle\rangle) + 1 \;=\; F(g)(n) + 1$$

a contradiction.

3 Recursion vs Tail Recursion

How does the ordinal $\|\Phi\|$ reflect the "complexity" of Φ ?

Definition. Associate with each terminating operator Φ the functional

$$H_\Phi \;:\; \mathrm{Tree}(\Phi) \;\to\; (N \to N) \;\to\; (N \to N)$$

defined by the following tail recursion over nodes σ on Tree(Φ) :

$$\begin{array}{lcll} H_\Phi\ (\sigma)(g)(m) & = & m & \text{if } \sigma \text{ is a leaf} \\ & = & H_\Phi\ (\sigma \frown g(m))(g)(m+1) & \text{otherwise.} \end{array}$$

Theorem 3.1 *For each terminating operator Φ we have*

$$H_\Phi\ (\langle n\rangle)(g)(0) \;=\; \mathit{least}\ y.\, T_\Phi(n, \bar{g}(y), y)\ ) \;.$$

Therefore

$$F_\Phi\ (g)(n) \;=\; U(\ H_\Phi\ (\langle n\rangle)(g)(0)\ ) \;.$$

Therefore $m \;=\; F_\Phi(g)(n)$ can be computed by the following abstract "while program" over Tree(Φ)

```
$\sigma$ := $\langle n\rangle$ ;
$m$ := 0 ;
while $\sigma$ is not a leaf do $\sigma$ := $\sigma \frown g(m)$ ; $m$ := $m+1$ od ;
$m$ := $U(m)$ .
```

Proof. Let $y_0 \;=\;$ least y. $T_\Phi(n, \bar{g}(y), y)$. Then from the definition above we compute

$$\begin{array}{lcl} H_\Phi\ (\langle n\rangle)(g)(0) & = & H_\Phi\ (\langle n, g(0)\rangle)(g)(1) \\ & = & H_\Phi\ (\langle n, g(0), g(1)\rangle)(g)(2) \\ & = & H_\Phi\ (\langle n, g(0), g(1), g(2)\rangle)(g)(3) \\ & = & \ldots \\ & = & H_\Phi\ (\langle n, g(0), \ldots, g(y_0-1)\rangle)(g)(y_0) \\ & = & y_0 \;. \end{array}$$

The rest follows immediately from the Normal Form for F_Φ and from the direct translation of the tail-recursive definition of H_Φ into a while-loop.

Definition. Call a terminating operator Φ, and its associated recursive functional F_Φ, *α-recursive* where α is an ordinal, if there is a function $|.| : N \to \alpha$ such that for all g and n, the recursive calls on f made by Φ in evaluating the right hand side of

$$f(n) \;=\; \Phi(g,f)(n)$$

are always of the form $f(m)$ where $|m| < |n|$.

Example. The previous example of an iteration operator Φ is ω-recursive since we can simply take $|n| = n$.

Lemma 3.2 *If Φ is α-recursive then $\|\Phi\| \leq \omega^\alpha$.*

Proof. Suppose f is defined from a given function g by α-recursion thus

$$f(n) \;=\; \Phi(g,f)(n)\,.$$

For any fixed n let $f(n_0)$, $f(n_1)$, ... , $f(n_i)$,... be the finitely-many recursive calls used in evaluating the right hand side, so $|n_i| < |n| < \alpha$ for each i. Note that in general there may be nested calls, meaning that the value of n_i might depend upon some other $f(n_j)$'s.

In the computation tree Tree(Φ), all of these recursive calls, and the successive calls needed for their evaluations etcetera, must be arranged linearly along the g-branch below node $\langle n\rangle$.

What we are looking for is a function ψ on ordinals such that

$$\|\langle n\rangle\| \;\leq\; \psi(|n|)\,.$$

But from the above comment we see that ψ must satisfy the condition :

$$\psi(|n|) \;\leq\; \Sigma_i \psi(|n_i|) \;\leq\; \max_i \psi(|n_i|)\cdot\omega$$

and so the obvious solution is

$$\psi(|n|) \;=\; \omega^{|n|}\,.$$

Therefore

$$\|\langle n\rangle\| \;\leq\; \omega^{|n|} \;<\; \omega^\alpha$$

and hence, taking the supremum over all nodes $\langle n\rangle$ in Tree(Φ), we obtain

$$\|\Phi\| \;\leq\; \omega^\alpha\,.$$

Note. If in the above Lemma, there were a fixed bound k on the number of recursive calls on f induced by Φ, then the ψ function need only satisfy :

$$\psi(|n|) \;\leq\; \psi(|n_0|) + \psi(|n_1|) + \ldots + \psi(|n_{k-1}|) \;\leq\; \max_i \psi(|n_i|)\cdot k$$

and so we could then choose a better bound :

$$\psi(|n|) = k^{|n|}$$

in which case

$$\|\Phi\| \leq k^{\alpha} .$$

Combining the Lemma with the previous Theorem, we thus have an "exponential trade-off" in the ordinal complexity of the transformation from general recursive programs to tail recursive ones. In fact the result below goes back originally to Tait(1961) who first showed how arbitrary nested recursions could be transformed into unnested ones. But it was re-examined more recently in Fairtlough-Wainer(1992) in the context of transformations to "while"-programs. The proofs there are different from Tait's, and different again from the ones presented here, although they all have the same germ..

Theorem 3.3 *Every α-recursive operator Φ can be transformed into an ω^{α}-tail-recursive definition of H_{Φ} such that for all $g : N \to N$ and all $n \in N$,*

$$F_{\Phi}\ (g)(n) \;=\; U(\ H_{\Phi}\ (\langle n\rangle)(g)(0)\ ) .$$

Furthermore, if there is a fixed bound k on the number of recursive calls in Φ, then ω^{α} can be strengthened to k^{α}.

Example. Let Φ be the operator defining a binary function $f = F_{\Phi}(g)$ as follows ;

$$\Phi(g, f)(n, m) = \textbf{if } n = 0 \textbf{ then } g(m) \textbf{ else } f(n-1, f(n-1, m)) .$$

Clearly Φ is ω-recursive under the assignment $|(n, m)| = n$ and, writing f_n to denote the function $\lambda m.f(n, m)$, we see that for all $n, m \in N$,

$$f_n\ (m) = f_{n-1} \circ f_{n-2} \circ \ldots \circ f_1 \circ f_0\ (\ g(m)\ ) .$$

Now for any strictly decreasing sequence of numbers $\sigma = \langle i_1, \ldots, i_r\rangle$ set

$$H(\sigma)(g) := f_{i_1} \circ f_{i_2} \circ \ldots \circ f_{i_r} .$$

Then for all n, m we have

$$f_n\ (m) = H\ (\langle n\rangle)(g)(m)$$

and furthermore, H is definable by the following tail recursion :

$$\begin{array}{lcl} H\ (\langle\rangle)(g)(m) & = & m \\ H\ (\sigma)(g)(m) & = & H\ (\tau)(g)(g(m)) \end{array}$$

where if $\sigma = \langle i_1, \ldots, i_r\rangle$ then

$$\tau = \langle i_1, \ldots, i_{r-1}, i_r - 1, i_r - 2, \ldots, 1, 0\rangle .$$

But notice that this tail recursive definition of H is a 2^ω-recursion under the assignment

$$|\sigma| \;=\; 2^{i_1} + 2^{i_2} + \ldots + 2^{i_r}\;.$$

Thus we have transformed the doubly-nested ω-recursion Φ into a 2^ω-tail- recursion.

Remark. Of course there is no difference *set theoretically* between the ordinals ω and 2^ω. However the structure of their respective well-orderings used in the above example is quite different. It is the increase in complexity of the well-ordering on 2^ω which reflects the cost in transforming the ω-recursive Φ to a tail recursion. In other words it is not really the set-theoretic notion of ordinal which matters here, but rather the "intensional" representation of it by means of a chosen well-ordering. This distinction is brought out further in the next section.

4 Majorization Hierarchies

As noted above, an ordinal for us is not simply a set. What matters is the way in which it is presented, for it is the presentation that provides a structure over which we can make recursive definitions. See Buchholz, Cichon and Weiermann (1994) for an alternative but closely related treatment of the ideas below.

Definitions. (i) By a *presentation* of a countable ordinal α we mean a chosen sequence of *finite* subsets $\alpha[i] \subset \alpha$ satisfying :

$$\alpha[0] \;\subset\; \alpha[1] \;\subset\; \ldots \;\subset\; \alpha[i] \;\subset\; \alpha[i+1] \;\subset\; \ldots$$

and

$$\alpha \;=\; \bigcup_i \alpha[i]\;.$$

We also insist that $0 \in \alpha[0]$ for all $\alpha > 0$.

(ii) Note that a presentation of α induces a presentation of each $\beta < \alpha$ by

$$\beta[i] \;=\; \alpha[i] \cap \beta\;.$$

(iii) Each presentation of α determines a sequence of predecessor functions $P_i : \alpha + 1 \rightarrow \alpha$ defined by

$$P_i(0) \;=\; 0 \;\text{ and }\; P_i(\beta) \;=\; \max\, \beta[i] \text{ if } 0 < \beta \leq \alpha\;.$$

Thus if k is the cardinality of $\beta[i]$ we have :

$$\beta[i] \;=\; \{0 = P_i^k(\beta) < P_i^{k-1}(\beta) < \ldots < P_i^2(\beta) < P_i(\beta)\}\;.$$

Definitions. Given fixed presentations of α and β, the predecessor functions P_i provide a natural way of recursively defining presentations of the sum, product and exponential as follows :

$$\begin{array}{lcl}
(\alpha+0)[i] & = & \alpha[i] \\
(\alpha+\beta)[i] & = & (\alpha+P_i(\beta))[i] \cup \{\alpha+P_i(\beta)\} \\
(\alpha\cdot 0)[i] & = & 0[i] \\
(\alpha\cdot\beta)[i] & = & (\alpha\cdot P_i(\beta)+\alpha)[i] \\
\alpha^0[i] & = & 1[i] \\
\alpha^\beta[i] & = & (\alpha^{P_i(\beta)}\cdot\alpha)[i]\ .
\end{array}$$

For $k \in N$ we choose the presentation $k[i] = \{0, 1, \ldots, k-1\}$ for every i. Then if we give ω the presentation $\omega[i] = (i+1)[i] = \{0, 1, \ldots, i\}$ the above definitions yield standard presentations for all ordinals below $\epsilon_0 = \sup\{1, \omega, \omega^\omega, \omega^{\omega^\omega}, \ldots\}$.

Definition. Let $F : (N \to N)^2 \to (N \to N)$ be any total functional, let α be any fixed countable ordinal with a chosen presentation, and let h, g be given functions. Then the *hierarchy* generated by iterating F over α consists of the functions $f_\beta : N \to N$ where $\beta \leq \alpha$, defined as follows :

$$\begin{array}{lcl}
f_0(n) & = & h(n) \\
f_\beta(n) & = & F(g, f_{P_n(\beta)})(n) \text{ if } 0 < \beta \leq \alpha\ .
\end{array}$$

Lemma 4.1 *Suppose the functional F satisfies the following "majorisation conditions" when restricted to strictly increasing, positive functions g and f :*

$$f(n) \ <\ F(g,f)(n) \ <\ F(g,f)(n+1)$$

and

$$\forall m \geq n\ (f(m) \leq f'(m)) \ \Rightarrow\ F(g,f)(n) \ \leq\ F(g,f')(n)\ .$$

Suppose also that h and g are strictly increasing and that g is positive.

Then the hierarchy $\{f_\beta : \beta \leq \alpha\}$ generated by F satisfies :

- $\gamma \in \beta[n] \ \Rightarrow\ f_\gamma(n) < f_\beta(n)$.
- *f_β is strictly increasing.*

Proof. By inductions over $\beta \leq \alpha$.

If $\gamma \in \beta[n]$ then either $\gamma \in P_n(\beta)[n]$ or $\gamma = P_n(\beta)$ and so by the induction hypothesis and the first majorisation condition we have

$$f_\gamma(n) \ \leq\ f_{P_n(\beta)}(n) \ <\ F(g, f_{P_n(\beta)})(n) \ =\ f_\beta(n)\ .$$

To see that f_β is strictly increasing, first notice that either $P_n(\beta) \in P_{n+1}(\beta)[n+1]$ or else $P_n(\beta) = P_{n+1}(\beta)$, because $\beta[n] \subset \beta[n+1]$. Therefore by the induction hypothesis we have

$$\forall m \geq n+1\ (f_{P_n(\beta)}(m) \leq f_{P_{n+1}(\beta)}(m))\ .$$

So by the induction hypothesis and the majorisation conditions,

$$\begin{aligned} f_\beta(n) &= F(g, f_{P_n(\beta)})(n) \\ &< F(g, f_{P_n(\beta)})(n+1) \\ &\leq F(g, f_{P_{n+1}(\beta)})(n+1) \\ &= f_\beta(n+1)\,. \end{aligned}$$

This completes the proof.

Majorisation Hierarchies. The following are the main examples of hierarchies generated by the above Lemma - and we shall make use of them later.

- Choosing $F(g,f) = g \circ f$ we obtain :

$$G_0(g)(n) \ = \ 0\ ; \ \ G_\beta(g)(n) \ = \ g(\ G_{P_n(\beta)}(g)(n)\ )\,.$$

- Choosing $F(g,f) = f \circ g$ we obtain :

$$H_0(g)(n) \ = \ n\ ; \ \ H_\beta(g)(n) \ = \ H_{P_n(\beta)}(g)(\ g(n)\ )\,.$$

- Choosing $F(g,f) = f \circ f = f^2$ we obtain :

$$B_0(g)(n) \ = \ g(n)\ ; \ \ B_\beta(g)(n) \ = \ B_{P_n(\beta)}(g)^2(n)\,.$$

These are called respectively the "Slow Growing", "Hardy" and (a version of the) "Fast Growing" hierarchies. When g is chosen to be the successor function we merely write $G_\beta(n)$, $H_\beta(n)$, $B_\beta(n)$ instead of $G_\beta(succ)(n)$, $H_\beta(succ)(n)$, $B_\beta(succ)(n)$. Although the definition of G does not quite satisfy the conditions of the Lemma, because we start with h the constant zero function, nevertheless the G functions will be strictly increasing after the first limit stage.

Lemma 4.2 *$G_\beta(n) \ = \ $ cardinality of $\beta[n]$.*

Proof. By a simple transfinite induction on β. If $\beta = 0$ then $\beta[n]$ is empty so its cardinality is $0 = G_0(n)$. If $\beta > 0$ then $\beta[n] = P_n(\beta)[n] \cup \{P_n(\beta)\}$ and so

$$\text{card}\,\beta[n] \ = \ 1 + \text{card}\,P_n(\beta)[n] \ = \ 1 + G_{P_n(\beta)}(n) \ = \ G_\beta(n)\,.$$

Lemma 4.3

$$H_{\gamma+\beta}(g) \ = \ H_\gamma(g) \ \circ \ H_\beta(g)$$

and

$$H_{2^\beta} \ = \ B_\beta\,.$$

Proof. (i) By induction on β. The first identitiy holds automatically if $\beta = 0$ since $H_0(g)$ is just the identity. If $\beta > 0$ then by the earlier definition of $\gamma + \beta[n]$ we see that $P_n(\gamma + \beta) = \gamma + P_n(\beta)$ and so

$$\begin{aligned} H_{\gamma+\beta}(g)(n) &= H_{P_n(\gamma+\beta)}(g)(g(n)) \\ &= H_{\gamma+P_n(\beta)}(g)(g(n)) \\ &= H_\gamma(g) \circ H_{P_n(\beta)}(g)\ (g(n)) \\ &= H_\gamma(g) \circ H_\beta(g)\ (n)\,. \end{aligned}$$

(ii) Again by induction on β. If $\beta = 0$ then

$$H_{2^0}(g)(n) \;=\; H_1(g)(n) \;=\; H_0(g)(g(n)) \;=\; g(n) \;=\; B_0(g)(n).$$

If $\beta > 0$ then by inspecting the definition of $2^\beta[n]$ given earlier we see that $P_n(2^\beta) = P_n(2^{P_n(\beta)} + 2^{P_n(\beta)})$, and therefore using the first part of the Lemma we have

$$\begin{aligned} H_{2^\beta}(g)(n) &= H_{P_n(2^{P_n(\beta)}+2^{P_n(\beta)})}(g)(g(n)) \\ &= H_{2^{P_n(\beta)}+2^{P_n(\beta)}}(g)(n) \\ &= H_{2^{P_n(\beta)}}(g) \circ H_{2^{P_n(\beta)}}(g)\ (n) \\ &= B_{P_n(\beta)}(g) \circ B_{P_n(\beta)}(g)\ (n) \\ &= B_\beta(g)(n) \quad \text{by definition of } B. \end{aligned}$$

Remark. As we shall see, the above relationship between the doubly nested definition of B and the tail recursive definition of H is in fact a cut elimination result.

5 Ordinal Bounds on Equational Derivations

This section is based on work of Fairtlough (1991), Fairtlough-Wainer (1992) and Handley-Wainer (1994). The idea is to assign uniform ordinal bounds to derivations of recursive functions in Kleene's Equation Calculus, from which we can read off sub- recursive hierarchy classifications of their computational complexity. (Complexity is viewed "in the large" - we are not concerned here with questions about feasible or polynomially bounded computation, but rather with general methods for comparing the complexities of natural classes of recursive definitions.)

Let E be a system of recursion equations (i.e. a recursive program) defining functions $f_0, \ldots, f_k$ from "given" functions $g = g_0, \ldots, g_l$ including the constant zero, the identity and the successor. Let t denote a variable-free term built up from numerals by repeated applications of the function symbols f_i and g_j. Write $t_0 \to_E t_1$ if t_0 rewrites to t_1 in one step by replacing any subterm $f_i(\vec{n})$ by its defining term in E. Let ordinal α be given with a fixed presentation.

Then the relation

$$n : N \vdash_E^\alpha t = m$$

meaning "from inputs less than or equal to n we can derive that t has value m within ordinal bound α", is generated inductively according to the following rules (we omit the "E" although the dependence on a given system of defining equations is to be understood) :

Ax $n : N \vdash^{\alpha} g(\vec{n}) = m$ if $\max \vec{n} \leq n$ and $g(\vec{n})$ has value m.

E $$\frac{n : N \vdash^{\beta} t_1 = m}{n : N \vdash^{\alpha} t_0 = m} \quad \text{if } t_0 \to_E t_1.$$

C $$\frac{n : N \vdash^{\beta_0} t_0 = k \quad \max(n,k) : N \vdash^{\beta_1} t_1(k) = m}{n : N \vdash^{\alpha} t_1(t_0) = m}.$$

R In the E and C rules the restriction is that $\beta, \beta_0, \beta_1 \in \alpha[n]$.

Remark. Of course any derivation of $n : N \vdash^{\alpha}_{E} t = m$ is finite, so why need we consider possibly infinite ordinal bounds ? The reason is that if f is a function defined by a system of equations E, then what we want to do is find a *uniform* bound α such that

$$\forall n \in N\, (f(n) = m \Rightarrow n : N \vdash^{\alpha}_{E} f(n) = m).$$

It is the recursion rule R which allows us to do this. Then α will provide an abstract measure of the complexity of the program defining f.

Lemma 5.1 *Suppose $n : N \vdash^{\alpha}_{E} t = m$. Let g now denote any fixed positive, strictly increasing unary function which bounds all the givens $g_0, \ldots, g_l$ in E, and let $g' = g + 1$. Then $m \leq B_\alpha(g)(n)$ and the height of the derivation tree is bounded by $B_\alpha(g')(n)$.*

Proof. By induction over the derivation of $n : N \vdash^{\alpha}_{E} t = m$ according to the Ax, E, and C rules restricted by R. The Ax and E rules are trivial since if $\max(\vec{n}) \leq n$ then for each $j \leq l$ we have, for $\beta \in \alpha[n]$,

$$g_j(\vec{n}) \leq g(n) \leq B_\beta(g)(n) < B_\alpha(g)(n) < B_\alpha(g')(n).$$

Now suppose $n : N \vdash^{\alpha} t = m$ arises by the C rule applied to premises

$$n : N \vdash^{\beta_0} t_0 = k \quad \text{and} \quad \max(n,k) : N \vdash^{\beta_1} t_1(k) = m\,.$$

Then by the induction hypothesis, and since $\beta_0, \beta_1 \in \alpha[n]$,

$$\max(n,k) \leq B_{\beta_0}(g)(n) \leq B_{P_n(\alpha)}(g)(n)$$

and

$$m \leq B_{\beta_1}(g)(\max(n,k)) \leq B_{P_n(\alpha)}(g)(\max(n,k))\,.$$

Therefore

$$m \leq B_{P_n(\alpha)}(g)(\, B_{P_n(\alpha)}(g)(n)\,) = B_\alpha(g)(n).$$

Furthermore the height of the derivation - viz. the least number greater than the heights of its two premises - is bounded by

$$\begin{aligned}
\max \quad & \{B_{\beta_0}(g')(n), B_{\beta_1}(g')(\max(n,k))\} + 1 \\
\leq \; & \max\{B_{\beta_0}(g')(n), B_{\beta_1}(g')(B_{P_n(\alpha)}(g)(n))\} + 1 \\
\leq \; & \max\{B_{P_n(\alpha)}(g')(n) + 1, B_{P_n(\alpha)}(g')(B_{P_n(\alpha)}(g)(n) + 1)\} \\
\leq \; & B_{P_n(\alpha)}(g')(\, B_{P_n(\alpha)}(g)(n) + 1 \,) \\
\leq \; & B_{P_n(\alpha)}(g')(\, B_{P_n(\alpha)}(g')(n) \,) \\
= \; & B_\alpha(g')(n) \, .
\end{aligned}$$

Definition. Let α be an infinite countable ordinal with a fixed chosen presentation. Then there is a bijection $\pi : \alpha \to N$ and so we can represent each $\beta < \alpha$ as a number $b = \pi(\beta)$ and each predecessor function P_n as a number-theoretic function p_n such that $p_n(b) = \pi(P_n(\beta))$. We can assume without loss of generality that $0 = \pi(0)$. Then we say that a function f is defined by α - *recursion* from the givens h and g if for some term T

$$\begin{aligned}
f(0,n) &= h(n) \\
f(b,n) &= T\,(g,f;n)
\end{aligned}$$

wherein each recursive call on f inside T is of the form $f(p_{t_0}(b), t_1)$ for certain subterms t_0 and t_1.

Lemma 5.2 *Suppose f is defined from the givens h, g by the α - recursion equations E as above. Then there is a fixed $k \in N$ such that for every $b = \pi(\beta)$ we have*

$$\forall n \in N \; (f(b,n) = m \;\Rightarrow\; n : N \;\vdash^{k.(\beta+1)}\; f(b,n) = m) \, .$$

Proof. Choose k to be the length (number of symbols) of the term T, plus 1. Proceed by induction on $\beta < \alpha$. The case $b = 0$ follows from the axiom $n : N \;\vdash^0\; h(n) = m$ by one application of the E rule, since $f(0,n) \to_E h(n)$ and $0 \in k[n]$ as long as $k > 0$. Now assume $b > 0$ and, by the induction hypothesis, that the result holds for each recursive call $f(p_{m_0}(b), m_1)$ used in evaluating $f(b,n) = T(g,f;n)$. We prove, by an inner induction over all subterms t of $T(g,f;n)$, that if t has length i and value m, then

$$n : N \;\vdash^{k.\beta+i}\; t = m \qquad *$$

The desired result then follows immediately for if t is the whole term T we have

$$n : N \;\vdash^{k.\beta+k-1}\; T = m$$

and hence $n : N \;\vdash^{k.(\beta+1)}\; f(b,n) = m$ follows by one further rewriting step.

To prove * consider three cases. Either t is n in which case * is just an axiom. Or t is of the form $g(t_0, \ldots, t_r)$ with g a "given", in which case * follows straightforwardly by $r+1$ applications of the C rule. Or else t is of the form $f(p_{t_0}(b), t_1)$ for certain subterms t_0 and t_1 where

$$n : N \;\vdash^{k.\beta+i_0}\; t_0 = m_0 \quad \text{and} \quad n : N \;\vdash^{k.\beta+i_1}\; t_1 = m_1 \, .$$

In this case, by the overall induction hypothesis on β, we have for the appropriate value m,

$$m_1 : N \ \vdash^{k.(P_{m_0}(\beta)+1)} \ f(p_{m_0}(b), m_1) = m$$

and since $k.(P_{m_0}(\beta)+1)[m_1] \subset k.\beta[\max(n, m_0, m_1)]$, we can also derive

$$\max(n, m_0, m_1) : N \ \vdash^{k.\beta} \ f(p_{m_0}(b), m_1) = m \ .$$

Therefore by two successive applications of the C rule we obtain first,

$$\max(n, m_0) : N \ \vdash^{k.\beta+i_1+1} \ f(p_{m_0}(b), t_1) = m$$

and then, since the length i of term t is at least $i_0 + i_1 + 1$,

$$n : N \ \vdash^{k.\beta+i} \ f(p_{t_0}(b), t_1) = m \ .$$

This completes the proof.

Definitions. Let $REC(< \tau)$ denote the class of all functions definable from the constant, addition and (modified) subtraction functions, by repeated applications of composition and α - recursion with $\alpha < \tau$.

Let $DER(< \tau)$ denote the class of all functions f for which there is a recursive program E (with constants, addition and subtraction as the only givens) and an ordinal $\alpha < \tau$ such that

$$\forall \vec{n} \in N^l \ (f(\vec{n}) = m \ \Rightarrow \ \max \vec{n} : N \ \vdash^{\alpha}_{E} \ f(\vec{n}) = m) \ .$$

Let $COMP(B_{<\tau})$ denote the class of all functions computable (in any reasonable model of computation) with resource bound B_α for some $\alpha < \tau$. Since B_ω is exponential the size of B_α, even for relatively small infinite α, already blurs any fine distinctions between choices of computational models.

Theorem 5.3 *Suppose τ is a limit ordinal closed under addition and $> \omega$. Suppose also that we have a fixed presentation of τ and a bijection $\pi : \tau \rightarrow N$ such that (i) for every $\beta < \tau$, $\beta \in \tau[\pi(\beta)]$, and (ii) the function $(n, b) \mapsto p_n(b)$ is primitive recursive. Then*

$$REC(< \tau) \ = \ \bigcup_k DER(< k \cdot \tau) \ = \ \bigcup_k COMP(B_{<k \cdot \tau}) \ .$$

Proof. (i) To show $REC(< \tau) \subset \bigcup_k DER(< k \cdot \tau)$ suppose for example that f is defined by an α - recursion from given functions g where $\alpha < \tau$ and where it is inductively assumed that $g \in DER(< k \cdot \gamma)$ for sufficiently large k. Then by Lemma 5.2 there is an appropriate k such that for all n and all $b = \pi(\beta)$ with $\beta < \alpha$,

$$n : N \ \vdash^{k.(\beta+1)} \ f(b, n) = m \ .$$

If we insert above the axioms introducing the givens g, their derivations with bound $k \cdot \gamma$, we obtain

$$n : N \ \vdash^{k.(\gamma+\beta+1)} \ f(b, n) = m$$

and hence

$$\max(b,n) : N \vdash^{k.(\gamma+\beta+1)} f(b,n) = m$$

where this is now an "outright" derivation from axioms defining only addition and subtraction. Now since $\beta \in \tau[b] \cap \alpha = \alpha[b]$ we have

$$k \cdot (\gamma + \beta) \in k \cdot (\gamma + \alpha)[b] \subset k \cdot (\gamma + \alpha)[\max(b,n)]$$

and hence

$$k \cdot (\gamma + \beta + 1)[\max(b,n)] \subset k \cdot (\gamma + \alpha)[\max(b,n)] .$$

Therefore we obtain for all $b, n \in N$,

$$\max(b,n) : N \vdash^{k.(\gamma+\alpha)} f(b,n) = m$$

and since τ is closed under addition, $\gamma + \alpha < \tau$, so f belongs to $DER(< k \cdot \tau)$.

(ii) To show $DER(< k \cdot \tau) \subset COMP(B_{<k\cdot\tau})$, suppose f is defined by a recursive program E (with only addition and subtraction as givens) in such a way that for some $\alpha < k \cdot \tau$,

$$\forall \vec{n}(f(\vec{n}) = m \Rightarrow \max \vec{n} : N \vdash^{\alpha} f(\vec{n}) = m) .$$

Then by Lemma 5.1, the finite height of the derivation tree for $f(\vec{n})$, and all the values m of subcomputations performed therein, are bounded by $B_\alpha(g)(\max \vec{n})$ for some linear function g. So for each input $\vec{n}$ the entire derivation tree for $f(\vec{n})$ can be encoded as a machine computation with resource bound exponential in $B_\alpha(g)$. Since $B_\omega(n) = B_\omega(succ)(n) = n + 2^{n+1}$, we have $g < B_{k\cdot\omega}$ provided $k > 0$, and hence $B_\alpha(g) < B_{k\cdot\omega+\alpha}$ and hence the resource bound in computing f will be less than $B_\omega \circ B_\alpha(g)$ which in turn is less than $B_{k\cdot\omega+\alpha} \circ B_{k\cdot\omega+\alpha} = B_{k\cdot\omega+\alpha+1}$. But $k \cdot \omega + \alpha + 1 < k \cdot \tau$ and hence $f \in COMP(B_{<k\cdot\tau})$.

(iii) To show $COMP(B_{<k\cdot\tau}) \subset REC(< \tau)$ we only need to prove that $B_{k\cdot\alpha} \in REC(< \tau)$ for every $\alpha < \tau$, because if f is computable within resource bound $B_{k\cdot\alpha}$ then f is definable from $B_{k\cdot\alpha}$ by compositions and ω - recursions only, and so f will also be in $REC(< \tau)$. But notice that

$$B_{k\cdot\alpha}(n) \;=\; B_{k\cdot P_n(\alpha)+k}(n) \;=\; B^{2^k}_{k\cdot P_n(\alpha)}(n)$$

the 2^k - times iterate of $B_{k\cdot P_n(\alpha)}$. So $B_{k\cdot\alpha}$ is definable by a 2^k - times nested α - recursion, and therefore it belongs to $REC(< \tau)$.

This completes the proof.

Examples. $REC(< \omega.2)$ is the class of all Csillag-Kalmar elementary functions, $REC(< \omega^2)$ is the class of all primitive recursive functions, and $REC(< \epsilon_0)$ is the class of all provably recursive functions of Peano Arithmetic.

Derivations of Tail Recursions. A *tail* α - recursion from givens h and g is a definition of the form

$$\begin{array}{lcl} f(0,n) & = & h(n) \\ f(b,n) & = & f(p_n(b), g(n)) \, . \end{array}$$

Following the idea of Lemma 5.2 we notice that the derivation of the recursion step uses only a simple form of the C rule where the left hand premise is an axiom :

$$\frac{\dfrac{n : N \vdash^{0} g(n) = m' \qquad m' : N \vdash^{2.P_n(\beta)} f(p_n(b), m') = m}{n : N \vdash^{2.P_n(\beta)+1} f(p_n(b), g(n)) = m}}{n : N \vdash^{2.\beta} f(b,n) = m} \, .$$

Thus if $\beta \in \alpha[\pi(\beta)]$ we can prove inductively that

$$\max(b,n) : N \vdash_0^{2.\alpha} f(b,n) = m$$

where $\vdash_0$ now denotes derivability using the Ax and E rules, but only these restricted forms of the C rule.

Definitions. (i) Let $TAIL(< \tau)$ denote the class of functions definable from constants, addition and subtraction by repeated compositions and *tail* α - recursions where $\alpha < \tau$.

(ii) Let $DER_0(< \tau)$ denote the class of all functions f for which there is a recursive program E (with constants, addition and subtraction as the only givens) and an ordinal $\alpha < \tau$ such that

$$\forall \vec{n} \in N^l \; (f(\vec{n}) = m \; \Rightarrow \; \max \vec{n} : N \vdash_0^{\alpha} f(\vec{n}) = m) \, .$$

Theorem 5.4 *Let τ satisfy the same conditions as in the previous Theorem. Then*

$$TAIL(< \tau) \;\; = \;\; \bigcup_k DER_0(< k \cdot \tau) \;\; = \;\; \bigcup_k COMP(H_{<k \cdot \tau}) \, .$$

Proof. The above comments are enough to show how the proof of the first part of Theorem 5.3 can be modified to yield $TAIL(< \tau) \subset DER_0(< k \cdot \tau)$, in fact with $k = 2$.

The proof of $DER_0(< k \cdot \tau) \subset COMP(H_{<k \cdot \tau})$ follows exactly the same lines as the proof of the second part of Theorem 5.3. But instead of using Lemma 5.1 we now need a corresponding lemma giving the appropriate bounding functions for $\vdash_0$ derivations. Since only a restricted form of the C rule is allowed here, a careful inspection of the proof of 5.1 shows that it is the H - functions which now give the appropriate bounds. For if we have a derivation

$$\frac{n : N \vdash^{0} g(n) = m' \qquad m' : N \vdash_0^{\beta} t(m') = m}{n : N \vdash_0^{\alpha} t(g(n)) = m}$$

where inductively we assume $m \leq H_\beta(g)(m')$ then since $\beta \in \alpha[n]$ we obtain

$$m \;\leq\; H_\beta(g)(g(n)) \;\leq\; H_{P_n(\alpha)}(g)(g(n)) \;=\; H_\alpha(g)(n)\,.$$

The proof of $COMP(H_{<k\cdot\tau}) \subset TAIL(<\tau)$ is again similar to the final part of the proof of 5.3. For any $\alpha < \tau$ it is easy to see that

$$H_{k\cdot\alpha}(g)(n) \;=\; H_{k\cdot P_n(\alpha)+k}(g)(n) \;=\; H_{k\cdot P_n(\alpha)}(g)(g^k(n))$$

and so $H_{k\cdot\alpha}(g)$ is definable by a tail α - recursion from the given function g^k. This completes the proof.

Theorem 5.5 *Let τ satisfy the same conditions as in the previous Theorems. Then for each k,*

$$DER(< k\cdot\tau) \;=\; DER_0(< 2^{k\cdot\tau})$$

and hence

$$REC(<\tau) \;=\; \bigcup_k TAIL(< 2^{k\cdot\tau})\,.$$

Proof. Simply recall that $B_\beta = H_{2^\beta}$ and apply the above Theorems.

Remark. In terms of equational derivation, this is a cut elimination result. The first part shows that the "call by value" cut rule C can be reduced to the restricted form

$$\frac{g(n) : N \;\vdash\; t(g(n)) = m}{n : N \;\vdash\; f(n) = m} \quad \text{if} \;\; f(n) \rightarrow_E t(g(n))$$

at the cost of an exponential increase in the ordinal presentation. And the second part shows that this cut reduction amounts to the transformation from general recursive to tail recursive programs.

This prompts a further question. Since tail-recursion corresponds to the cut rule in which the left hand premise is an axiom, what kind of recursion is modelled by the cut rule in which the right hand premise is an axiom ?

Derivations of Pointwise Recursions. A *pointwise* α - recursion from givens h and g is a definition of the form

$$\begin{array}{lcl} f(0,n) & = & h(n) \\ f(b,n) & = & g(f(p_n(b),n))\,. \end{array}$$

Notice that the parameter n remains fixed in the recursive call on f and so the derivation of the recursion step uses only a C rule where now the right hand premise is an axiom :

$$\dfrac{\dfrac{n : N \;\vdash^{2.P_n(\beta)}\; f(p_n(b),n) = k \qquad k : N \;\vdash^0\; g(k) = m}{n : N \;\vdash^{2.P_n(\beta)+1}\; g(f(p_n(b),n)) = m}}{n : N \;\vdash^{2.\beta}\; f(b,n) = m}\,.$$

Thus if $\beta \in \alpha[b]$ where $b = \pi(\beta)$ we can prove inductively that

$$\max(b,n) : N \ \models^{2.\alpha} \ f(b,n) = m$$

where $\models$ now denotes derivability using the Ax and E rules, but only these right-hand restricted forms of the C rule. Since the ordinal bounds on this derivation of $f(b,n)$ all now lie in the finite set $2.\alpha[\max(b,n)]$ of cardinality $G_{2.\alpha}(\max(b,n))$, it is clear that the complexity of $\models$ -derivations will be bounded by the Slow Growing hierarchy. Thus for appropriate τ 's satisfying the additional condition that for each pair $\alpha_0, \alpha_1 < \tau$ there is an $\alpha < \tau$ such that $G_{\alpha_0} \circ G_{\alpha_1}$ is bounded by G_α, we have

$$POINTWISE(<\tau) \ = \ COMP(G_{<\tau}) \, .$$

Comparisons between the G and B hierarchies, and hence between pointwise recursion and arbitrary nested recursion, are more subtle than the exponential trade-off between B and H. However from Girard (1981) and others subsequently (see Wainer (1989) for direct computations) we have a map $\alpha \mapsto \alpha^+$ from ordinal presentations to ordinal presentations, defined at least on the initial segment below the proof theoretic ordinal of $(\Pi^1_1 - CA)_0$, such that

$$B_\alpha \ = \ G_{\alpha^+} \, .$$

Consequently for appropriate τ 's we obtain

$$REC(<\tau) \ = \ POINTWISE(<\tau^+) \, .$$

Examples. For the primitive recursive functions we have $\tau = \omega^2$ and $\tau^+ =$ (a presentation of) the first primitive recursively closed ordinal. For the provably recursive functions of arithmetic we have $\tau = \epsilon_0$ and $\tau^+ =$ the Howard ordinal. See e.g. Schwichtenberg-Wainer (1995).

The significance of pointwise recursion is that it is intimately connected with term-rewriting. The work of Cichon (1992), Weiermann (1991,1993,1995) and Handley-Wainer (1994) shows how the τ^+ ordinal gives a "termination ordering" along which the $< \tau$ - recursive functions are computable by rewriting. Thus the + - map and the associated cut elimination from $\vdash$ to $\models$ reflect a trade-off between derivations of "call-by-value" recursion on one hand, and computability by rewriting on the other.

References

(1) W. Buchholz, E.A. Cichon and A. Weiermann, A Uniform Approach to Fundamental Sequences and Subrecursive Hierarchies, Math. Logic Quarterly Vol. 40, 1994, pp 273 - 286.

(2) E.A. Cichon, Termination Proofs and Complexity Characterizations, in Proof Theory, Eds. P. Aczel, H. Simmons, S. Wainer, Cambridge 1992, pp 173 - 193.

(3) M.V. Fairtlough, Ordinal Complexity of Recursive Programs and their Termination Proofs, Ph.D. dissertation Leeds University, 1991.

(4) M.V. Fairtlough and S.S. Wainer, Ordinal Complexity of Recursive Definitions, Information and Computation Vol. 99, 1992, pp 123 - 153.

(5) W.G. Handley and S.S. Wainer, Equational Derivation versus Computation, Annals of Pure and Applied Logic Vol. 70, 1994, pp 17 - 49.

(6) S.C. Kleene, Extension of an Effectively Generated Class of Functions by Enumeration, Colloquium Mathematicum Vol. 6, 1958, pp 67 - 78.

(7) H. Schwichtenberg and S.S. Wainer, Ordinal Bounds for Programs, in Feasible Math. II, Eds. P. Clote, J. Remmel, Birkhäuser 1995, pp 387 - 406.

(8) W.W. Tait, Nested Recursion, Math. Annalen Vol. 143,1961, pp 236-250.

(9) S.S. Wainer, Slow Growing versus Fast Growing, Journ. Symbolic Logic Vol. 54, 1989, pp 608 - 614.

(10) A. Weiermann, Proving Termination for Term Rewriting Systems, in Computer Science Logic, Eds. E. Börger, G. Jäger, H. Kleine-Büning, M. Richter, Springer Lect. Notes in Computer Science Vol. 626, 1991, pp 419 - 428.

(11) A. Weiermann, Bounding Derivation Lengths with Functions from the Slow Growing Hierarchy, preprint Univ. Münster, 1993.

(12) A. Weiermann, Termination Proofs by Lexicographic Path Orderings yield Multiply Recursive Derivation Lengths, Theor. Computer Science, 1995, to appear.

Feasibly Categorical Models

Douglas Cenzer[1] and Jeffrey B. Remmel[2]

[1] University of Florida, Gainesville, FL 32611-2802, USA
[2] University of California at San Diego, La Jolla, CA 92093, USA

Abstract. We define a notion of a Scott family of formulas for a feasible model and give various conditions on a Scott family which imply that two models with the same family are feasibly isomorphic. For example, if A and B possess a common strongly p-time Scott family and both have universe $\{1\}^*$, then they are p-time isomorphic. These results are applied to the study of permutation structures, linear orderings, equivalence relations, and Abelian groups. For example, conditions on two permutation structures (A, f) and (B, g) are given which imply that (A, f) and (B, g) are p-time isomorphic.

Introduction

The focus of feasible mathematics has been on determining the complexity of certain classes of models. The classic example is the graph-coloring problem, where it is known that the family of finite graphs which can be 3-colored is a complete NP class. The problem of feasible colorings of infinite graphs was studied by the authors in [4]. Polynomial time model theory or more generally feasible model theory, the subject of this paper, considers the complexity of the model itself. Let $\omega = \{0, 1, \ldots\}$ denote the set of natural numbers. Let $\langle , \rangle$ denote some fixed recursive pairing function which maps $\omega \times \omega$ onto ω. Let $\phi_{e,n}$ denote the n-ary partial function on $(\{0,1\}^*)^n$ computed by the e-th Turing machine. Then we say that a structure $\mathcal{A} = (A, \{R_i^A\}_{i\in S}, \{f_i^A\}_{i\in T}, \{c_i^A\}_{i\in U})$, (where the universe A of $\mathcal{A}$ is a subset of $\{0,1\}^*$) is *recursive* if A is a recursive subset of $\{0,1\}^*$, S,T, and U are initial segments of ω, the set of relations $\{R_i^A\}_{i\in S}$ is uniformly recursive in the sense that there is a recursive function G such that for all $i \in S$, $G(i) = \langle n_i, e_i\rangle$ where R_i^A is an n_i-ary relation and ϕ_{e_i,n_i} computes the charateristic function of R_i^A, the set of functions $\{f_i^A\}_{i\in T}$ is uniformly recursive in the sense that there is a recursive function F such that for all $i \in T$, $F(i) = \langle n_i, e_i\rangle$ where f_i^A is an n_i-ary function and ϕ_{e_i,n_i} restricted to A^n computes f_i^A, and there is a recursive function interpreting the constant symbols in the sense that there is a recursive function H such that for all $i \in U$, $H(i) = c_i^A$. Note that if $\mathcal{A}$ is a recursive structure, then the atomic diagram of $\mathcal{A}$ is recursive. We say that a recursive structure $\mathcal{A} = (A, \{R_i^A\}_{i\in S}, \{f_i^A\}_{i\in T}, \{c_i^A\}_{i\in U})$, is *polynomial time* if A is a polynomial time subset of $\{0,1\}^*$ and the set of relations $\{R_i^A\}_{i\in S}$ and the set of functions $\{f_i^A\}_{i\in T}$ are uniformly polynomial time in the sense that, in addition of the functions G and F defined above, there are recursive functions G' and F' such that for $i \in S$, $G'(i) = m_i$ where for all $(x_1, \ldots, x_{n_i})$ in $(\{0,1\}^*)^{n_i}$, it takes at most $(max\{2, |x_1|, \ldots, |x_{n_i}|\})^{m_i}$ steps to compute $\phi_{e_i,n_i}(x_1, \ldots, x_{n_i})$

and for all $i \in T$, $G'(i) = q_i$ where for all $(x_1, \ldots, x_{n_i})$ in $(\{0,1\}^*)^{n_i}$, it takes at most $(max\{2, |x_1|, \ldots, |x_{n_i}|\})^{q_i}$ steps to compute $\phi_{e_i,n_i}(x_1, \ldots, x_{n_i})$. Note if $\mathcal{A}$ is polynomial time structure with infinitely many relation symbols or with infinitely many function symbols, then our definition of a polynomial time structure does not ensure that the atomic diagram of $\mathcal{A}$ is polynomial time. Thus we say $\mathcal{A}$ is uniformly polynomial time if the atomic diagram of $\mathcal{A}$ is polynomial time. Note that the fact that $\mathcal{A}$ is uniformly polynomial time implies, among other things, that the sequence of run times $\{x^{m_i} : i \in S\}$ and $\{x^{q_i} : i \in T\}$ are bounded by some fixed polynomial. Of course, if $\mathcal{A}$ is a structure over a finite language, then $\mathcal{A}$ is a polynomial time structure iff $\mathcal{A}$ is uniformly polynomial time structure.

There are two basic types of questions which have been studied in polynomial time model theory. First, there is the basic existence problem, i.e. is a given infinite recursive structure $\mathcal{A}$ isomorphic or recursively isomorphic to a polynomial time model. For example, the authors showed in [1] that every recursive relational structure is recursively isomorphic to a polynomial time model and that the standard model of arithmetic $(\omega, +, -, \cdot, <, 2^x)$ with addition, subtraction, multiplication, order and the 1-place exponential function is isomorphic to a polynomial time model. A more restricted kind of existence question is whether a given recursive model is isomorphic or recursively isomorphic to a polynomial time model which has a standard universe such as the binary representation of the natural numbers, $Bin(\omega)$, or the tally representation of the natural numbers, $Tal(\omega) = \{1^n : n \in \omega\}$. For example, Grigorieff [8] proved that every recursive linear ordering is isomorphic to a linear time linear ordering which has universe $Bin(\omega)$ while Cenzer and Remmel [1] showed that there exists a recursive copy of the linear ordering $\omega + \omega^*$ which is not recursively isomorphic to any polynomial time linear ordering which has universe $Bin(\omega)$. Here $\omega + \omega^*$ is the ordering obtained by taking a copy of $\omega = \{0, 1, 2, \ldots\}$ under the usual ordering followed by a copy the negative integers under the usual ordering. The general problem of determining which recursive models are isomorphic or recursively isomorphic to feasible models has been studied by the authors in [1], [2], and [5]. For example, it was shown in [2] that any recursive torsion Abelian group G is isomorphic to a polynomial time group A and that if the orders of the elements of G are bounded, then A may be taken to have a standard domain, i.e. either $Bin(\omega)$ or $Tal(\omega)$. Feasible linear orderings were studied by Grigorieff [8], by Cenzer and Remmel [1], and by Remmel [14] [16]. Feasible vector spaces were studied by Nerode and Remmel in [10] and [12]. Feasible Boolean algebras were studied by Cenzer and Remmel in [1] and by Nerode and Remmel in [11]. Feasible permutation structures and feasible Abelian groups were studied by Cenzer and Remmel in [2] and [5].

The second basic type of problem studied in polynomial time model theory is the problem of feasible categoricity. Here we say that a recursive model $\mathcal{A}$ is *recursively categorical* if any other recursive model isomorphic to $\mathcal{A}$ is in fact recursively isomorphic to $\mathcal{A}$. Defining a natural analogue of feasible categoricity is complicated by the fact that unlike the case of infinite recursive models, where any two infinite recursive universes are recursively isomorphic, it is not the case that any two polynomial time universes are polynomial time isomorphic. For example, $Bin(\omega)$ is not polynomial time isomorphic to $Tal(\omega)$. It turns out to be more natural to define polynomial categorical structures with respect to a fixed universe. Thus we say that a p-time structure $\mathcal{A}$ with universe $D \subseteq \{0,1\}^*$ is *p-time categorical with respect to D* if every p-time structure $\mathcal{B}$ with universe D which is isomorphic to $\mathcal{A}$ is necessarily p-time isomorphic to $\mathcal{A}$, i.e. there exist polynomial time functions f, g such that f restricted to D is an isomorphism from $\mathcal{A}$ onto $\mathcal{B}$ and g restricted to D is an isomorphism from $\mathcal{B}$ onto $\mathcal{A}$. For example, the problem of feasible categoricity for permutation structures and torsion Abelian groups was studied by Cenzer and Remmel in [5]. A permutation structure is a structure (A, f) where $f : A \to A$ is a permutation of A. The following results on poynomial time categoricity, and other notions of computable categoricity, of permutation structures were proved in [5].

(1) A recursive permutation structure is recursively categorical if and only if it has only finitely many infinite orbits.
(2) If the recursive permutation structure $\mathcal{A}$ is not recursively categorical, then there exist p-time structures $\mathcal{B}_1$ and $\mathcal{B}_2$, each isomorphic to $\mathcal{A}$, having the same standard universe (either $Bin(\omega)$ or $Tal(\omega)$), which are not recursively isomorphic to each other.
(3) If the recursive permutation structure $\mathcal{A}$ has an infinite orbit, or has infinitely many orbits of size q, for some finite q, and infinitely many other orbits, then there exist p-time structures $\mathcal{B}_1$ and $\mathcal{B}_2$, each isomorphic to $\mathcal{A}$, having the same standard universe (either $Bin(\omega)$ or $Tal(\omega)$), which are not primitive recursively isomorphic to each other.
(4) If $\mathcal{A} = (A, f)$ and $\mathcal{B} = (B, g)$ are two p-time permutation structures having only finite orbits and all but finitely many orbits have the same finite size q, then $\mathcal{A}$ and $\mathcal{B}$ are p-time isomorphic if $B = Tal(\omega)$ and are exponential time isomorphic if $B = Bin(\omega)$.

We note that Remmel [14] and Dzgoev [7] proved that a recursive linear ordering L is recursively categorical iff L has finitely many successivities, i.e. iff L has only finitely many pairs $x <_L y$ where y is an immediate successor of x in L. Remmel showed in [16] that there is in fact no p-time categoricity for linear orderings. That is, for any p-time linear ordering $L = (B, <)$ with standard universe $B = Bin(\omega)$ or $B = Tal(\omega)$, Remmel proved the following.

(5) If L has infinitely many successivities, then there exists a p-time ordering L' with universe B which is isomorphic to L but not recursively isomorphic to L.
(6) If L has only finitely many successivities, then there exists a p-time ordering L' with universe B which is isomorphic to L but not primitive recursively isomorphic to L.

These results are typical in the sense that they show that polynomial time categorical structures over a standard universe are relatively rare and that polynomial time categoricity can fail in quite strong ways. The existence questions and categoricity questions discussed above for polynomial time models can easily be generalized to other notions of feasible models such as exponential time, polynomial or exponential space, etc. The goal of this paper is to develop general conditions under which some form of feasible categoricity can be demonstrated. The reader is refered to [9] for basic notions of complexity theory and to [5] or [16] for basic notions of feasible model theory.

1 Feasibly categorical structures

The specific purpose of this paper is to develop syntactic approximations of the notion of feasible categoricity. Nurtazin [13] and Goncharov [6] provided sufficient conditions to ensure that a model $\mathcal{A}$ with universe A is recursively categorical, namely if there is a finite sequence $(c_0, \dots, c_{k-1})$ of elements of A and a recursive sequence (called a *Scott family*) of recursive existential formulas $\{\phi_n(x_1, \dots, x_{m-1}, c_0, \dots, c_{k-1}) : n < \omega\}$ in the extended language with names for $c_0, \dots, c_{k-1}$ satisfying the following two conditions:

(1) Every $a_0, \dots, a_{m-1} \in A$ satisfies one of the formulas ϕ_n;
(2) For each n and for any two sequences $(a_0, \dots, a_{m-1})$ and $(d_0, \dots, d_{m-1})$, if $\mathcal{A}$ satisfies $\phi_n(a_0, \dots, a_{m-1}, c_0, \dots, c_{k-1})$ and $\phi_n(d_0, \dots, d_{m-1}, c_0, \dots, c_{k-1})$, then $(A, a_0, \dots, a_{m-1}, c_0, \dots, c_{k-1})$ is isomorphic to $(A, d_0, \dots, d_{m-1}, c_0, \dots, c_{k-1})$ via the map which sends a_i to d_i for $i < m$ and c_i to c_i for $i < k$.

There are three observations about polynomial time (p-time) models which affect the notion of categoricity. First, there are p-time subsets of $\{0,1\}^*$ which are not p-time isomorphic, for example $Bin(\omega)$ and $Tal(\omega)$. Thus even the unadorned model $\mathcal{A}$ with no relations or functions has p-time models which are not p-time isomorphic. In fact, for any p-time relational structure $\mathcal{A}$, there is a p-time structure $\mathcal{B}$ which is recursively isomorphic, but not even primitive recursively isomorphic to $\mathcal{A}$, see [5]. Thus we consider categoricity with respect to a fixed universe, such as $Tal(\omega)$ or $Bin(\omega)$. Second, the iteration of a polynomial time algorithm does not produce a p-time function. For example, if $f(0) = 2$ and $f(n+1) = f(n)^r$, then $f(n) = 2^{r^n}$. Therefore, we often need to relax the complexity of the isomorphism between p-time structures. For example, if $\mathcal{A}$ and $\mathcal{B}$ are two p-time groups both isomorphic to the direct sum of infintely many cyclic groups of order p, then $\mathcal{A}$ and $\mathcal{B}$ are EXPTIME isomorphic if both have universe $Tal(\omega)$ and are double-exponential time isomorphic if both have universe $Bin(\omega)$, but are not necessarily p-time isomorphic in either case, see [5]. Third, we must consider the notion of **honest witnesses.** In a p-time model, the existence of an element a such that $\phi(a, x_0, \dots, x_{m-1})$ does not guarantee that we can compute such an element even in primitive recursive time. Thus, for example, there are two p-time models, both having universe $Tal(\omega)$, of the simple structure $\mathcal{A} = (A, R)$, where R is an infinite, co-infinite subset of A, which are not even primitive recursively isomorphic, see [5]. These considerations lead us to define various feasible analogues of a Scott family. For example, a Scott family

$\{\phi_n(x_1,\ldots,x_{m-1},c_0,c_1,\ldots,c_{k-1}) : n < \omega\}$, for a p-time model $\mathcal{A}$ with universe A, satisfying (1) and (2) as described above is said to be *strongly p-time* if there is some fixed integer $r > 1$ such that the following conditions are satisfied, for each $m > 0$.

(3) For any finite sequence $a_0,\ldots,a_{m-1}$ of elements of $\mathcal{A}$, we can compute in time $\leq (max\{2,m,|a_0|,\ldots,|a_{m-1}|\})^r$ a formula ϕ_t from the list such that $\phi_t(a_0,\ldots,a_{m-1},c_0,c_1,\ldots,c_{k-1})$ holds in $\mathcal{A}$.

(4) For each formula $\phi_t(x_0,\ldots,x_{m-1},x_m,c_0,c_1,\ldots,c_{k-1})$ and each $a_0,\ldots,a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0,\ldots,a_{m-1},a,c_0,c_1,\ldots,c_{k-1})$, then there exists such an a with $|a| \leq (m+2)^r + max\{|a_0|,\ldots,|a_{m-1}|\}$.

(5) For each $\phi_t(x_0,\ldots,x_{m-1},x_m,c_0,c_1,\ldots,c_{k-1})$ and each $a_0,\ldots,a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0,\ldots,a_{m-1},a,c_0,c_1,\ldots,c_{k-1})$, then we can compute an a as described in (4) in time $\leq (max\{2,m,|a_0|,\ldots,|a_{m-1}|\})^r$.

Theorem 1. *If $\mathcal{A}$ and $\mathcal{B}$ possess a common strongly p-time Scott family, then $\mathcal{A}$ and $\mathcal{B}$ are p-time isomorphic if both have universe $Tal(\omega)$ and are exponential time isomorphic if both have universe $Bin(\omega)$.*

Proof. For notational convenience, we shall give the proof when the Scott family has no parameters as the proof for Scott families with parameters is essentially the same. Also for any $a \in \omega$, we shall abuse notation and simply write a for the tally representation of a when the universe is $Tal(\omega)$ and write a for the binary representation of a when the universe is $Bin(\omega)$. The isomorphism ϕ between $\mathcal{A}$ and $\mathcal{B}$ is given by the usual back-and-forth construction. At stage n, we will have defined finite sequences $a_0,\ldots,a_{n-1}$ and $b_0,\ldots,b_{n-1}$ such that the map taking a_i to b_i is an isomorphism from the restriction of $\mathcal{A}$ to $\{a_0,\ldots,a_{n-1}\}$ to the restriction of $\mathcal{B}$ to $\{b_0,\ldots,b_{n-1}\}$. We begin at stage 1 with the following procedure. Let $a_0 = 0$. Then compute the formula ϕ_t such that $\mathcal{A}$ satisfies $\phi_t(a_0)$. Next, we compute $b \in B$ such that $\mathcal{B}$ satisfies $\phi_t(b)$ and we let $b_0 = b$. At stage 2, there are two choices. If $b_0 = 0$, then $b_1 = 1$ and if $b_0 > 0$, then $b_1 = 0$. Then we compute the formula ϕ_t such that $\phi_t(b_0,b_1)$ holds in $\mathcal{B}$. Next, we compute $a \in A$ such that $\phi_t(a_0,a)$ holds in $\mathcal{A}$ and we let $a_1 = a$. In general, at stage $2m+1$, we let a_{2m} be the least a not equal to a_i for any $i < 2m$. Then we compute the formula ϕ_t such that $\phi_t(a_0,\ldots,a_{2m})$ holds in $\mathcal{A}$. Next, we compute $b = b_{2m}$ such that $\phi_t(b_0,b_1,\ldots,b_{2m-1},b)$ holds in $\mathcal{B}$. At stage $2m+2$, we similarly let b_{2m+1} be the least b not equal to b_i for any $i < 2m+1$, we compute the formula ϕ_t such that $\phi_t(b_0,\ldots,b_{2m+1})$ holds in $\mathcal{B}$ and we then compute $a = a_{2m+1}$ such that $\phi_t(a_0,\ldots,a_{2m},a)$ holds in $\mathcal{A}$. It is easy to see by induction that, for each m,

$$a_{2m} \leq 2m, \quad b_{2m+1} \leq 2m+1 \tag{1}$$

$$|a_{2m+1}| \leq 1 + 3^r + 5^r + \cdots + (2m+3)^r \leq (2m+5)^{r+1}, \tag{2}$$

$$\text{and } |b_{2m}| \leq 2^r + 4^r + \cdots + (2m+2)^r \leq (2m+4)^{r+1}. \tag{3}$$

We can now find an upper bound for the time required for each stage of the computation. At stage $2m+1$, given $a_0,\ldots,a_{2m-1}$ and $b_0,\ldots,b_{2m-1}$, we require time $\leq cm$ to compute a_{2m}, for some constant c. Next, we require time $\leq (2m+4)^{r(r+1)}$ to find the required formula ϕ_t. Then we require time $\leq (2m+2)^r + (2m+2)^{r+1}$ to compute b_{2m}. At stage $2m+2$, we require time cm to compute b_{2m+1}, then time $\leq (2m+4)^{r(r+1)}$ to find the formula ϕ_t, and finally time $\leq (2m+3)^r + (2m+4)^{r+1}$ to compute a_{2m+1}. Thus the total time required for each stage n is bounded by kn^{r^2+r} for some fixed constant k. It follows that the total time required for stages $1, 2, \ldots, m+1$ is bounded by dm^{r^2+r+1} for some fixed constant d. Then in order to compute $\phi(a) = b$, we must follow the construction until we find $a = a_m$ and then let $b = b_m$. Now we know from the construction that $a = a_m$ for some $m \leq 2a$. Thus, in tally, we need to compute no more than $2a$ stages, so that we can compute $\phi(a)$ in time $\leq d(2a)^{r^2+r+1}$. Since in tally $|a| = a$ (for $a > 0$), it follows that ϕ is polynomial time. A similar argument shows that ϕ^{-1} is also p-time. In binary, we only have $a \leq 2^{|a|}$, so that the first $2a$ stages of the construction require time

$$d(2a)^{r^2+r+1} \leq d2^{r^2+r+1}2^{(r^2+r+1)|a|}. \tag{4}$$

It follows that ϕ is exponential time and similarly that ϕ^{-1} is also exponential time. □

We apply this theorem to various models to obtain the following results.

Corollary 2. *Let $\mathcal{A} = (Tal(\omega), f)$ and $\mathcal{B} = (Tal(\omega), g)$ be two isomorphic p-time permutation structures such that for some fixed integer k,*
(i) for any a and a' in the same orbit,

$$|a'| \leq |a| + k \tag{5}$$

and
(ii) for any $a_0, a_1, \ldots, a_{m-1} \in B$ and any finite q, if there is an orbit of size q not containing any of $a_0, \ldots, a_{m-1}$, then there is such an orbit containing an element a of size

$$|a| \leq max\{|a_0|, \ldots, |a_{m-1}|\} + (m+2)^k. \tag{6}$$

Then $\mathcal{A}$ and $\mathcal{B}$ are p-time isomorphic.

Proof. First observe that the sizes of the orbits are bounded by k. The Scott family of formulas for these models may be described as follows. For each pair (x, y) of variables, there are $2k$ *basic* formulas, $f^i(x) = y$ and $\neg f^i(x) = y$ for $i = 0, \ldots, k-1$. Then for any finite sequence $x_0, \ldots, x_{m-1}$ of variables, there are $2km^2$ basic formulas $\pm f^i(x_h) = x_j$. A formula is in the Scott family if it is the conjunction of such $2km^2$ basic formulas $\bigwedge_{h,i,j} \phi_{h,i,j}$, where each $\phi_{h,i,j}$ is $\pm f^i(x_h) = x_j$. Given a finite sequence $a_0, \ldots, a_{m-1}$ of elements of $\mathcal{A}$, we obtain the formula $\phi(x_0, \ldots, x_{m-1})$ satisfied by $a_0, \ldots, a_{m-1}$ by computing $f^i(a_h)$ for all $i < k$ and comparing the result to each a_j. Let

$$n = max\{2, m, a_0, \ldots, a_{m-1}\} \tag{7}$$

and suppose that $f(x)$ can be computed in time $\leq |x|^c$ for some c. It follows that, for each i and j, $|f^i(a_j)| \leq |a_j|^{c^i}$ and that $f^i(a_j)$ may be computed in time

$$t \leq |a_j|^c + |a_j|^{c^2} + \cdots + |a_j|^{c^i} \leq n^{c^k}. \tag{8}$$

Thus we may compute a list of all the $f^i(a_j)$ in total time

$$t \leq kmn^{c^k} \leq n^{c^k+k+1}. \tag{9}$$

For each of the km^2 pairs of basic formulas $\pm f^i(x_h) = x_j$, we then compare $f^i(a_h)$ with a_j to determine which formula is true. This can be done in time $\leq dn$ for sufficiently large d, so that all of the comparisons can be done in total time $\leq dkm^2 \leq n^{2+d+k}$. The formula $\phi = \phi(x_0, \ldots, x_{m-1})$ satisfied by $a_0, \ldots, a_{m-1}$ is simply the conjunction of these basic formulas. Thus ϕ can be computed in time

$$t \leq n^r \quad \text{where } r = dk + 2k + 4, \tag{10}$$

so that clause (3) in the definition of a strongly p-time Scott family is satisfied. Now suppose that $\phi(a_0, \ldots, a_{m-1}, a)$ is true for some a, where ϕ is one of the formulas in the Scott family. There are two cases. First, we may have $a = f^i(a_j)$ for some i, j. Thus, by (i),

$$|a| \leq |a_j| + k. \tag{11}$$

Second, we may have a in some orbit of size q disjoint from the orbits of $a_0, \ldots, a_{m-1}$. Then by (ii), there is some b with

$$|b| \leq max\{|a_0|, \ldots, |a_{m-1}|\} + (m+2)^k \tag{12}$$

with a similar property. It follows from the definition of the Scott family that $\phi(a_0, \ldots, a_{m-1}, b)$. Thus clause (4) is satisfied. The computation of the element a is likewise in two cases. If $a = f^i(a_j)$ for some i and j, then this computation can be done in time $\leq |a_j|^{c^i}$ as indicated above. If $a \neq f^i(a_j)$ for any i and j, then a must belong to a new orbit and we can read the size q of that orbit from ϕ. Then we search for the least b with

$$|b| \leq max\{|a_0|, \ldots, |a_{m-1}|\} + (m+2)^k \tag{13}$$

which is different from every $f^i(a_j)$ and such that $f^q(b) = b$ and $f^i(b) \neq b$ for any $i < q$. We may assume by the argument above that we have already computed the orbits of each of the elements $a_0, \ldots, a_{m-1}$. Then the choices for b are limited to

$$b \leq n + (m+2)^k \leq n + (2n)^k \leq n^{2k+1} \tag{14}$$

and each choice has length $\leq n^{2k+1}$. Also, there are $\leq mk$ values $f^i(a_j)$, each of length $\leq n + k$ to compare b with. Then for each b and each $i, j < k$, the comparison of b with $f^i(a_j)$ takes time

$$t \leq (n+k)^2 \leq n^{k+2}. \tag{15}$$

Since there are $(mk)^2$ such tasks, the total time required for this comparison is

$$t \leq m^2k^2n^{k+2}. \tag{16}$$

It is also necessary to compute $f^i(b)$ for $i \leq q \leq k$ and $b < n^{2k+1}$ and to check that $f^i(b) \neq b$ for $i < q$ and that $f^q(b) = b$. Each of these computations take at most $(n^{(2k+1)})^{c^q} = n^{c^q(2k+1)}$ steps and the comparisons take time at most $n^{(4k+2)c^q}$. It is now easily verified that clause (5) is satisfied. □

We note that this result may be applied to permutation structures with finitely many finite orbit sizes. For example, we can construct a model $(Tal(\omega), f)$ satisfying the hypothesis of Corollary 2 which has infinitely many orbits of size 2 and infinitely many orbits of size 3. (Just alternate the orbit sizes, so that $f(x) = x-1$ if $x = 1(mod\ 5)$, $f(x) = x - 2$ if $x = 4(mod\ 5)$ and $f(x) = x + 1$ otherwise.) The restrictions (i) and (ii) are needed, since it is shown in [5] that there are p-time models with universe $Tal(\omega)$ isomorphic to $(Tal(\omega), f)$ but not primitive recursively isomorphic to $(Tal(\omega), f)$. On the other hand, if all but finitely many orbits have the *same* finite size q, then it is shown that even without conditions (i) and (ii), any p-time model with universe $Tal(\omega)$ is actually p-time isomorphic to $\mathcal{A}$. Our next result was proved directly by the second author in [16]. To state that result, we need to define when a polynomial time dense linear order without endpoints possesses quasi-real time density functions. A p-time linear order $\mathcal{A} = (A, <^A)$ is said to have *quasi-real time* density functions if there exist functions f, g, and h and a constant c such that for all $a, b \in A$ with $a <^A b$:

(i) $f(a)$ may be computed in time $\leq |a| + c$ and $f(a) <^A a$,
(ii) $g(a)$ may be computed in time $\leq |a| + c$ and $a <^A g(a)$, and
(iii) $h(a, b)$ may be computed in time $\leq max\{|a|, |b|\} + c$ and $a <^A h(a, b) <^A b$.

If the run time bounds in clauses (i), (ii), (iii) are replaced respectively by $\leq c \cdot |a|$, $\leq c \cdot |a|$, and $\leq c \cdot max\{|a|, |b|\}$, then we say that f, g, and h are linear time density functions of for $\mathcal{A}$. Similarly if the run time bounds in clauses (i), (ii), (iii) are replaced respectively by $\leq (max\{2, |a|\})^c$, $\leq (max\{2, |a|\})^c$, and $\leq (max\{2, |a|, |b|\})^c$, then we say that f, g, and h are polynomial time density functions of for $\mathcal{A}$.

Corollary 3. *Let $\mathcal{A} = (Bin(\omega), \leq^A)$ and $\mathcal{B} = (Bin(\omega), \leq^B)$ be two dense p-time linear orderings without end point and possessing quasi-real time density functions. Then $\mathcal{A}$ and $\mathcal{B}$ are exponential time isomorphic.*

Proof. We will outline the argument. The formulas in the Scott family are conjunctions of the basic formulas $x_i < x_j$, $x_i = x_j$. Each formula gives a complete description of the ordering (with equalities) of the variables involved. We compute the formula satisfied by a sequence $a_0, \ldots, a_{m-1}$ of elements of $\mathcal{A}$, simply by comparing each pair of elements. It is easy to see that clause (3) is satisfied. Any formula $\phi_t(x_0, \ldots, x_{m-1}, x_m)$ describes the ordering on $x_0, \ldots, x_{m-1}, x_m$, so that it must say one of four things about the position of x_m. Either $x_m = x_i$ for some i, $x_m < x_i$ where $x_i \leq x_j$ for all $j \leq m$, $x_m > x_i$ where $x_i \geq x_j$ for all $j \leq m$, or $x_i < x_m < x_j$ for some pair i, j such that x_j is the successor of x_i in the ordering on $\{x_0, \ldots, x_{m-1}\}$. The quasi-real time density functions f, g, h clearly suffice to demonstrate clause (4) as well as clause (5). □

The conditions given in Theorem 1 are quite strong. Thus we consider next a weaker version. The Scott family $\{\phi_n(x_1, \ldots, x_{m-1}, c_0, \ldots, c_{k-1}) : n < \omega\}$ for a polynomial time model $\mathcal{A}$ with universe A satisfying (1) and (2) is said to be *strongly exponential time* if there is some fixed integer $r > 1$ such that the following conditions are satisfied, for each $m > 0$.

(3)′ For any finite sequence $a_0, \ldots, a_{m-1}$ of elements of $\mathcal{A}$, we can compute in time $\leq r^m(max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r$ a formula ϕ_t from the list such that $\phi_t(a_0, \ldots, a_{m-1}, c_0, \ldots, c_{k-1})$ holds in $\mathcal{A}$.

(4)′ For each $\phi_t(x_0, \ldots, x_{m-1}, x_m, c_0, \ldots, c_{k-1})$ and each $a_0, \ldots, a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0, \ldots, a_{m-1}, a, c_0, \ldots, c_{k-1})$, then there exists such an a with $|a| \leq r \cdot max\{r^m, |a_0|, \ldots, |a_{m-1}|\}$.

(5)′ For each $\phi_t(x_0, \ldots, x_{m-1}, x_m, c_0, \ldots, c_{k-1})$ and each $a_0, \ldots, a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0, \ldots, a_{m-1}, a, c_0, \ldots, c_{k-1})$, then we can compute an a as described in (4)′ in time $\leq r^m(max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r$.

Theorem 4. *If $\mathcal{A}$ and $\mathcal{B}$ possess a common strongly exponential time Scott family, then $\mathcal{A}$ and $\mathcal{B}$ are exponential time isomorphic if both have universe $Tal(\omega)$ and are double exponential time isomorphic if both have universe $Bin(\omega)$.*

Proof. Again we give the proof only in the case of a exponential time Scott family with no parameters as the proof in the case with parameters is essentially the same. The isomorphism ϕ between $\mathcal{A}$ and $\mathcal{B}$ is given by the back-and-forth construction described in the proof of Theorem 1. It is easy to see by induction that, for each m,

$$a_{2m} \leq 2m, \quad b_{2m+1} \leq 2m+1, \quad |a_{2m+1}| \leq r^{2m+2}, \quad \text{and } |b_{2m}| \leq r^{2m+1}. \tag{17}$$

It follows that, at stage $m+1$, we can compute the desired formula ϕ_t satisfied by $(a_0, \ldots, a_m)$ or by $(b_0, \ldots, b_m)$ in time

$$t \leq r^{m+1} r^{(m+1)r} = (r^{r+1})^{m+1} \tag{18}$$

and we can then find the desired element b_m or a_m in a similarly bounded time. Thus the total time t required for stages $1, 2, \ldots, m+1$ satisfies

$$t \leq d(r^{r+1})^{m+2} \tag{19}$$

for some fixed constant d. It is now easy to see as in the proof of Theorem 1 that the isomorphism can be computed in exponential time in tally and in double exponential time in binary. □

We note that condition $(3)'$ in Theorem 4 can be replaced by the stronger condition $(3)''$ that the list of all formulas having m variables in the Scott family is of length $\leq r^m$ and can be computed in exponential time from 1^m and that each the statisfaction in $\mathcal{A}$ of each $\phi_t(a_0, \ldots, a_{m-1}, c_0, \ldots, c_{k-1})$ in the Scott family can be tested in time

$$t \leq (max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r. \tag{20}$$

(Thus we can find the formula ϕ_t by trying all formulas in the list.) For a tally universe, condition $(5)'$ then follows from conditions $(3)''$ and $(4)'$, i.e. the element a can be found by testing the satisfaction in $\mathcal{A}$ for all elements b where $b \leq r^m(max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r$ of the formulas of $m+1$ variables $\phi_t(a_0, \ldots, a_{m-1}, b, c_0, \ldots, c_{k-1})$ in the Scott family. For a binary universe, there are 2^{r^m} elements of length r^m, so that this search takes too long. Thus if we replace condition $(3)'$ by $(3)''$ and replace condition $(4)'$ by condition (4) so that the possible length of a_m is exponential in m, then $(5)''$ would follow from $(3)''$ and (4) since the search to be done in double exponential time. The following is a result that can be proved using Theorem 4 in much the same way that Corollary 2 was proved from Theorem 1.

Corollary 5. *Let $\mathcal{A} = (B, \equiv^A)$ and $\mathcal{B} = (B, \equiv^B)$ be two polynomial time models of an equivalence relation $\equiv$ such that, for some fixed integer k, both models satisfy the following:*
(i) for any a and a' in the same equivalence class,

$$|a|' \leq k \cdot |a| \text{ if } B = Tal(\omega) \tag{21}$$

or

$$|a|' \leq k + |a| \text{ if } B = Bin(\omega) \tag{22}$$

and
(ii) for any $a_0, \ldots, a_{m-1} \in B$ and any finite q, if there is an equivalence class of size q not containing any of $a_0, \ldots, a_{m-1}$, then there is such a class containing an element b of size

$$|b| \leq r \cdot max\{r^m, |a_0|, \ldots, |a_{m-1}|\} \text{ if } B = Tal(\omega) \tag{23}$$

or

$$|b| \leq (m+2)^k + max\{a_0|, \ldots, |a_{m-1}|\} \text{ if } B = Bin(\omega). \tag{24}$$

Then $\mathcal{A}$ and $\mathcal{B}$ are exponential time isomorphic if $B = Tal(\omega)$, and double exponential time isomorphic if $B = Bin(\omega)$.

The following result, which was proved directly in [16], follows from Theorem 4 by the same argument by which Corollary 3 was proved from Theorem 1.

Corollary 6. *Let $\mathcal{A} = (B, \leq^A)$ and $\mathcal{B} = (B, \leq^B)$ be two dense p-time linear orderings without endpoints and possessing linear time density functions. Then $\mathcal{A}$ and $\mathcal{B}$ are exponential time isomorphic if $B = Tal(\omega)$ and double exponential time isomorphic if $B = Bin(\omega)$.*

Next we give a slight modification of Theorem 4. Let us say that a Scott family is *strongly EXPTIME* if it satisfies a slight modification of the definition of a strongly exponential time family, that is, we replace clause (4)′ by the following clause:

(4)″ For each $\phi_t(x_0, \ldots, x_{m-1}, x_m, c_0, \ldots, c_{k-1})$ and each $a_0, \ldots, a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0, \ldots, a_{m-1}, a, c_0, \ldots, c_{k-1})$, then there exists such an a with $|a| \leq r^m \cdot max\{r, |a_0|, \ldots, |a_{m-1}|\}$.

Theorem 7. *If $\mathcal{A}$ and $\mathcal{B}$ possess a strongly EXPTIME Scott family, then $\mathcal{A}$ and $\mathcal{B}$ are EXPTIME isomorphic if both have universe $Tal(\omega)$ and are double exponential time isomorphic if both have universe $Bin(\omega)$.*

Proof. The modification in the proof is that the size of a_m and b_m is now of order r^{m^2}, which makes the computation EXPTIME rather than exponential time. □

This theorem can be applied to obtain a result on torsion Abelian groups. Let $o(a)$ be the order of a in a group G. Let $G(a_0, \ldots, a_{m-1})$ be the subgroup of G generated by $a_0, \ldots, a_{m-1}$.

Corollary 8. *Let $\mathcal{A}$ and $\mathcal{B}$ be two isomorphic p-time torsion Abelian groups with the same universe $Tal(\omega)$ such that for some fixed integer k,*
(i) for any a, b,

$$|a + b| \leq k \cdot max\{|a|, |b|\} \tag{25}$$

and
(ii) for any $a_0, \ldots, a_{m-1}$ in either $\mathcal{A}$ or $\mathcal{B}$ and any finite q, if there is an element of order q not in $G(a_0, \ldots, a_{m-1})$, then there is such an element b of size

$$|b| \leq k^m \cdot max\{|a_0|, \ldots, |a_{m-1}|\}. \tag{26}$$

Then $\mathcal{A}$ and $\mathcal{B}$ are EXPTIME isomorphic if $B = Tal(\omega)$ and are double exponential time isomorphic if $B = Bin(\omega)$.

Proof. The formula $\phi_t(a_0, \ldots, a_m)$ simply states which of the linear combinations $e_0a_0 + \cdots + e_ma_m = 0$; where $e_i \in Z(q)$. Since there are q^m such formulas, condition (3)′ is clearly satisfied. For condition (4)″, we observe that if $a \in G(a_0, \ldots, a_{m-1})$, then we can compute a with fewer than qm additions, so that

$$|a| \leq k^{mq} \cdot max\{|a_0|, \ldots, |a_{m-1}|\}. \tag{27}$$

There are two cases in the verification of condition (5)′. If $a \in G(a_0, \ldots, a_{m-1})$, then we can compute a as above in on the order of $k^{mq} \cdot max\{|a_0|, \ldots, |a_{m-1}|\}$ steps. If a is independent of $a_0, \ldots, a_{m-1}$, then we apply clause (*ii*) and simply test all values for a with $a \leq k^m \cdot max\{|a_0|, \ldots, |a_{m-1}|\}$ □

We give one more general result. A Scott family $\{\phi_n(x_1, \ldots, x_{m-1}, c_0, \ldots, c_{k-1}) : n < \omega\}$ for a polynomial time model $\mathcal{A}$ with universe A satifying (1) and (2) as described above is said to be *polynomial time* if there is some fixed integer $r > 1$ such that the following conditions are satisfied, for each $m > 0$.

$(3)^p$ For any finite sequence $a_0, \ldots, a_{m-1}$ of elements of $\mathcal{A}$, we can compute in time $\leq (max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r$ a formula ϕ_t from the list such that $\phi_t(a_0, \ldots, a_{m-1}, c_0, \ldots, c_{k-1})$ holds in $\mathcal{A}$.

$(4)^p$ For each $\phi_t(x_0, \ldots, x_{m-1}, x_m, c_0, \ldots, c_{k-1})$ and each $a_0, \ldots, a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0, \ldots, a_{m-1}, a, c_0, \ldots, c_{k-1})$, then there exists such an a with $|a| \leq (max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r$.

$(5)^p$ For each $\phi_t(x_0, \ldots, x_{m-1}, x_m, c_0, \ldots, c_{k-1})$ and each $a_0, \ldots, a_{m-1} \in A$, if there exists a such that $\mathcal{A}$ satisfies $\phi_t(a_0, \ldots, a_{m-1}, a, c_0, \ldots, c_{k-1})$, then we can compute such an a in time $\leq (max\{2, m, |a_0|, \ldots, |a_{m-1}|\})^r$.

Theorem 9. *If $\mathcal{A}$ and $\mathcal{B}$ possess a p-time Scott family, then $\mathcal{A}$ and $\mathcal{B}$ are double exponential time isomorphic if both have universe $Tal(\omega)$ and are triple exponential time isomorphic if both have universe $Bin(\omega)$.*

The following corollary of Theorem 9 was proved directly in [16].

Corollary 10. *Let $\mathcal{A} = (B, \leq^A)$ and $\mathcal{B} = (B, \leq^B)$ be two dense p-time linear orderings without endpoints and possessing polynomial time density functions. Then $\mathcal{A}$ and $\mathcal{B}$ are double exponential time isomorphic if $B = Tal(\omega)$ and triple exponential time isomorphic if $B = Bin(\omega)$.*

References

1. Cenzer,D., Remmel, J.: Polynomial-time versus recursive models. Ann. Pure and Appl. Logic **54** (1991) 17–58
2. Cenzer,D., Remmel, J.: Polynomial-time Abelian groups. Ann. Pure and Appl. Logic **56** (1992) 313–363
3. Cenzer,D., Remmel, J.: Recursively presented games and strategies. Math. Social Sciences **24** (1992) 117–139
4. Cenzer,D., Remmel, J.: Feasible graphs and colorings. Math. Logic Quarterly, (to appear)
5. Cenzer,D., Remmel, J.: Feasibly categorical abelian groups, in "Feasible Mathematics II" editors P. Clote and J. Remmel, Prog. in Comp. Science and Appl. Logic **13**, Birkhäuser (1995) 91–154
6. Goncharov, S.S.: Autostability and computable families of constructivization. Algebra and Logic **14** (1975) 392–409
7. Goncharov, S.S., Dzgoev, V.D.: Autostability of models. Algebra and Logic **19** (1980) 28–37
8. Grigorieff, S.: Every Recursive linear ordering has a copy in DTIME(n). J. Symbolic Logic **55** (1990) 260–276
9. Hopcroft, J., Ullman, J.: "Formal Languages and Their Relations to Automata". Addison Wesley (1969)
10. Nerode, A., Remmel, J.: Complexity theoretic algebra I, vector spaces over finite fields, in "Proceedings of Structure in Complexity, 2d Annual Conference", Computer Science Press (1987) 218–239
11. Nerode, A., Remmel, J.: Complexity theoretic algebra II, the free Boolean algebra. Ann. Pure and Applied Logic **44** (1989) 71–99

12. Nerode, A., Remmel, J.: Complexity theoretic algebra: vector space bases, in "Feasible Mathematics", editors S. Buss and P. Scott, Prog. in Comp. Science and Appl. Logic **9** Birkhäuser (1990) 293–319
13. Nurtazin, A.: "Completable classes and algebraic conditions for autostability". Ph. D. thesis, Novosibirsk (1974)
14. Remmel, J.: Recursively categorical linear orderings. Proc. Amer. Math. Soc. **83** (1981) 387–391
15. Remmel, J.: When is every recursive linear ordering of type μ recursively isomorphic to a p-time linear order over the binary representation of the natural numbers?, in "Feasible Mathematics", editors S. Buss and P. Scott, Prog. in Comp. Science and Appl. Logic **9** Birkhäuser (1990) 321–341
16. Remmel, J. Polynomial-time categoricity and linear orderings, in "Logical Methods", editors J. Crossley, J. Remmel, R. Shore, and M. Sweedler, Prog. in Comp. Science and Appl. Logic **12** Birkhäuser (1993) 321–341

Metafinite Model Theory

Erich Grädel* and Yuri Gurevich**

Abstract. Motivated by computer science challenges, we suggest to extend the approach and methods of finite model theory beyond finite structures.

Table of Contents

* Lehrgebiet Mathematische Grundlagen der Informatik, RWTH Aachen, D-52056 Aachen, Germany, graedel@informatik.rwth-aachen.de

** EECS Department, University of Michigan, Ann Arbor, MI 48109-2122, USA, gurevich@umich.edu. Partially supported by NSF grant CCR 92-04742 and ONR grant N00014-94-1-1182.

1 Finite models and beyond

Although questions involving finite structures have always been of interest to logicians, finite model theory has emerged as a separate research area only in the 1970's and early 1980's. Part of the motivation came from applications in computer science, in particular from databases and complexity theory. As was pointed out in [20], finite structures pose a nontrivial challenge for mathematical logic, in particular for model theory. Being closely related to the foundations of mathematics, classical logic is preoccupied with infinity. In fact most important classical results and techniques of mathematical logic (such as compactness, completeness, the usual preservation theorems) fail when only finite structures are considered. It was suggested in [20] that logicians should systematically develop a model theory of finite structures that is able to cope with the challenges from computer science.

Even though we believe that finite model theory has been rather successful, the time has come to re-examine the situation. Motivated again by challenges from computer science we feel that the strict adherence to finiteness is too restrictive and suggest to *extend the approach and methods of finite model theory beyond finite models.*

1.1 Motivation

Many of the finite objects appearing in computer science refer at least implicitly to infinite structures. In particular, this is the case with *objects that consist of both structures and numbers,* like e.g. graphs with weights on the edges. Such objects arise in many areas of mathematics and computer science, e.g. in optimization, databases, complexity theory and combinatorics. Although a single such object may be representable by a finite structure, it is not always desirable to do so. The numbers appearing in it live in an infinite structured domain, e.g. the field of reals or the arithmetic of natural numbers, and the arithmetical operations that we want to perform on these numbers may take us out of any *a priori* fixed finite subdomain. Thus it is desirable to work directly on the infinite structure, but to adjust the logical languages in an appropriate way so that certain complications coming from the infinity of the structure are avoided.

Databases. To explain the challenge of going beyond finite models and integrating structures and numbers, we first look at database theory, a particularly important area for such an approach. We refer to the books [1, 47] and the survey article [34] for background on database theory.

The common practice of viewing (a state of) a relational database as a finite structure is not always adequate; we are not the first to say that (see Sect. 1.4 in this connection). Let us look a little closer at the relationship between databases and finite model theory. In fact, database theory doesn't start with identifying relational databases and finite relational structures. Informally, a relational database is a finite collection of relations, each of which is a finite subset $R \subseteq D_1 \times \cdots \times D_m$ of tuples in a cartesian product of domains D_i;

the domains need not be finite, in fact it is often assumed that all domains are *countably infinite*. The *active domain* of the database is the set of those domain elements that appear in some relation. Since the relations are finite, so is the active domain. So actually, a database is a countably infinite structure all whose relations are finite. By considering the substructure induced by the active domain, a finite structure is obtained carrying all the relevant information. For many theoretical considerations one can forget at this point where the domain elements came from, and work with the finite structure instead.

However, in real databases some of the domains are not just plain sets, but themselves are (infinite) mathematical structures, e.g., the natural numbers with arithmetic. Traditionally the relations and functions structuring these domains are not considered as parts of the database; supposedly they are imposed "from outside". But of course, this additional structure of the domains is used in database applications. Commercial query languages like SQL have arithmetical operations and comparisons, as well as so-called *aggregate functions* like mean, sum, max, min that are applicable to the appropriate domains. In this case the restriction to the active domain is no longer convincing, since arithmetical operations may produce new numbers that were not previously stored in the database.

We thus believe that a more realistic logical approach to databases should be *systematically* developed, that does not adhere to the strict finiteness condition, but retains the essential achievements of finite model theory.

Discrete dynamic systems. Databases evolve in time and can be viewed as special discrete dynamic systems. Additional examples are ubiquitous in computer science: micro-processes, operating systems, compilers, programming languages, communication protocols. Discrete dynamic systems play an enormous rôle in computer science and engineering. The problem of formal specification of discrete dynamic system is very important and attracts much attention. In practice, the most popular approaches to the specification problem are operational approaches which formalize states of discrete dynamic systems in one form or another. For a logician, it is natural to formalize states as structures of first-order logic. This venue has been pursued in the evolving algebra approach; the venue is quite practical and fruitful [22].

Since states are finite they can be formalized as finite structures. However, it turns out that often it is more convenient and practical to incorporate various background structures into states and deal with infinite states. This is a rule, rather than an exception, in the evolving algebra literature (see [8]). Here we restrict ourselves to one simple example.

Imagine that a state of interest includes a stack of some objects which may be popped or pushed during the transition to the next state. There are many ways to implement a stack. Respectively there are many ways to represent a stack in a finite structure. But you may want to avoid excessive detailization, for example to make your verification proof simpler and cleaner. One solution is to have an auxiliary infinite universe of stacks with built-in pop and push operations and a nullary function that gives the stack of interest to us. The details of this simple example are explained in the EA Tutorial mentioned in [8]. More involved variations of the example appear in many places, in particular in Jim Huggins' correctness proof of the Kermit communication protocol (also referred to in [22]) where stacks are replaced by queues.

1.2 Metafinite structures

Logics with counting. There are logics, studied in the framework of finite model theory, that go some way towards integrating logic and arithmetic. These are the *logics with counting*, augmenting familiar logics like first-order logic or fixed-point logic with the ability to count the number of tuples in any definable relation. Syntactically this can be done by either *counting terms* or *counting quantifiers.*

The motivation for considering these logics comes from the observation that from the point of view of expressiveness, first-order logic (FO for brevity) has two main deficiencies: It has *no mechanism for recursion or unbounded iteration*, and it *cannot count.* There are several well-studied logics and database query languages that add recursion in one way or another to FO (or part of it), notably the various forms of fixed point logic, the query language Datalog and its extensions.

On ordered finite structures, some of these languages express precisely the queries that are computable in PTIME or other complexity classes. However, this is not the case for classes of arbitrary (not necessarily ordered) structures, and most of the known counterexamples involve counting. Thus, Immerman [29] proposed to add counting quantifiers to fixed point logic and asked whether this would suffice to capture PTIME. Although Cai, Fürer and Immerman [9] eventually answered this question negatively, fixed point logic with counting turned out to be an important logic, defining a natural level of expressiveness below PTIME, with a number of equivalent characterizations [17].

Logics with counting are two-sorted. With a one-sorted finite structure $\mathfrak{A}$ with universe A, one associates the two-sorted structure $\mathfrak{A}^* := (\mathfrak{A}, \mathfrak{N})$ where $\mathfrak{N} = (\{0, \dots, n\}, <))$ for $n = |A|$ and the canonical ordering $<$ on $\{0, \dots, n\}$. The two sorts are related by *counting terms* of the form $\#_x[\varphi]$ taking values in the second, numerical sort. The interpretation of $\#_x[\varphi]$ is the number of first-sort elements a that satisfy $\varphi(a)$. *(Inflationary) fixed point logic with counting* (FP + C) and *partial fixed point logic with counting* (PFP + C) are defined by closing first-order logic under counting terms and the usual rules for building inflationary or partial fixed points. The predicates defined by fixed point operators may be mixed, i.e. range over both sorts. We refer to [17, 31, 40, 41] and to Sect. 4.3 and 5.3 below for more background and results on fixed point logics with counting.

It should be noted, that although the second, numerical sort is of rather restricted form — just a linear ordering — this suffices to define any polynomial-time computable numerical function in fixed point logic. Thus it makes no difference if the numerical sort has additional relations and functions, e.g. modular addition and multiplication, as long as these are polynomial-time computable.

Here we will consider similar two-sorted structures with the following essential differences:

- The numerical sort need not be finite.
- The structures may contain functions from the first to the second sort.
- We consider more general operations than counting.

Metafinite structures. Answering the challenge to extend the approach and methods of finite model theory beyond finite models and integrating structures and numbers, we propose here a more general class of structures, which we call *metafinite structures*, and a number of logics to reason about them. Typical metafinite structures consist of *(i)* a primary part, which is a finite structure, *(ii)* a secondary part, which may be finite or infinite, and *(iii)* a set of "weight" functions from the first part into the second. Here is an example: a graph, the set of natural numbers with the usual arithmetical operations, and a weight function from the vertices (or the edges) of the graph to the natural numbers.

By itself, the notion of metafinite structures may seem an old hat. Indeed, they are just a special kind of two-sorted structures. Rather than just in the structures themselves, the novelty of our approach is primarily in the logics for such structures, which access the primary and the secondary part in different ways.

The term "metafinite structure" is loose; in most cases the secondary part will be an infinite numerical domain, so the structures are in fact perfectly infinite. The term "metafinite" reflects our intention to apply the approach and methods of finite model theory to these structures. In fact the infinity that we seek is very modest. It should not manifest itself too obtrusively, deviating our attention to phenomena that are pertinent to infinite structures only. Therefore our logics of metafinite structures — appropriate modifications of the usual logics of interest to finite model theory, such as first-order logic, fixed point logics or $L^{\omega}_{\infty\omega}$ — access the infinite part only in a limited way, for instance without variables (and therefore without quantifiers) over the secondary part. An important feature of these languages is that they contain, besides formulae and terms in the usual sense, a calculus of functions from the primary to the secondary part, which we call *weights*.

Encoding problems. Of course one may object that also a weighted structure, which consists of both a structure and a collection of numbers, can be encoded either by a pure structure or by binary string. This is true, but not always satisfactory.

To encode a graph with weights on edges by a unweighted graph one could, for instance, replace every edge (u, v) of weight w by w distinct nodes, each of them connected to u and v but to no other nodes. While the graph obtained in this way contains all information about the original weighted graph, it is very inconvenient to perform arithmetical computations on the encoded weights.

On the other side, encoding a structure (with or without weights) as a binary string requires that we order the structure and thus forces us to deal with presentations of structures which contradicts the spirit of the relational database approach.

1.3 Potential applications

We have mentioned databases and discrete dynamic systems as motivations of metafinite model theory. There are numerous other areas where this approach may be useful. We intended also to write a section on applications of metafinite model theory but his has to be deferred to a later paper. Instead we mention a few things here.

Optimization. Many important optimization problems are NP-hard and thus cannot be efficiently solved, unless P = NP. One way to cope with such problems is the design of *approximation algorithms* which do not necessarily find optimal solutions, but approximate ones, in the sense that the quality or cost of the produced solution differs from an optimal one only by constant factor. In fact many optimization problems admit efficient approximation algorithms, whereas for others it has been shown that also finding approximate solutions is NP-hard.

Papadimitriou and Yannakakis [42] set forth a new, logical approach for studying the approximation properties of optimization properties. Exploiting Fagin's logical characterization of NP by existential second-order logic, they introduced two syntactically defined classes of maximization problems, MAX SNP and MAX NP, and proved that all problems in these classes admit efficient approximation algorithms. The work of Papadimitriou and Yannakakis also was one of the starting points for a number of spectacular non-approximability results. In particular, the characterization of NP in terms of *probabilistically checkable proofs*, obtained by Arora et. al. [5], implies that no MAX SNP-hard problem can have a polynomial-time approximation scheme, unless P = NP.

Many optimization problems that appear in practice take structures with weights as input instances, e.g. graphs with one or more weight functions assigning numbers to vertices or edges. Important examples are the TRAVELLING SALESMAN PROBLEM, MAX FLOW/MIN CUT, most scheduling problems, and so on (see [13] for additional examples).

As mentioned already in [42] the result of Papadimitriou and Yannakakis can be extended to problems with weights. However, the weighted versions of MAX SNP and MAX NP as defined in [42] use the weights only in a rather limited way. We believe that metafinite structures provide the right framework to extend this approach to a more general definability theory of optimization problems with weights.

Numerical invariants of structures. In many branches of mathematics, functions that assign numerical parameters to mathematical structures play an important rôle. For instance, a large part of graph theory is devoted to the study of numerical invariants of graphs, such as genus, chromatic number, clique number, diameter, girth, etc. Metafinite model theory provides a framework for studying definability issues of numerical invariants and relating them, for instance, to computational complexity.

Fault-tolerance of queries. Suppose we have a relational database where every entry has some probability of being incorrect. What is the probability that the result of a given query is correct? What is the expectation for the "relative difference" of the query applied to the observed database with respect to the "actual database".

Such questions also involve objects consisting of a finite structure and a collection of numbers. An *unreliable database* can be defined as a pair $(\mathfrak{A}, \mu)$ consisting of a finite structure $\mathfrak{A}$ and a probability function μ that assigns to each each atomic or negated atomic fact a probability of being wrong. With $(\mathfrak{A}, \mu)$ we can associate a probability space of databases $\mathfrak{B}$ with probabilities $\nu(\mathfrak{B})$ to be understood as the probability that the 'actual' database is $\mathfrak{B}$.

Given a query Q against an unreliable database $(\mathfrak{A}, \mu)$, it is interesting to determine its *fault-tolerance.* For a Boolean query, the fault-tolerance is just the probability that the evaluation against the observed database $\mathfrak{A}$ gives the correct answer for the actual database $\mathfrak{B}$. For queries of positive arity, the fault-tolerance is defined to be proportional to the expected Hamming distance of $Q(\mathfrak{A})$ and $Q(\mathfrak{B})$, i.e. the expected number of tuples that distinguish between $Q(\mathfrak{A})$ and $Q(\mathfrak{B})$.

In Sect. 3.3 we will show how to address these questions in the framework of metafinite model theory. In particular, we will prove that the fault-tolerance of a conjunctive query is first-order definable.

Note that we can also consider unreliable metafinite databases. This gives examples where the secondary part has itself several sorts, namely one or more sorts for the numbers appearing in the database, and one sort over the real interval $[0, 1]$ for the error probabilities.

Computations over the real numbers. Blum, Shub, and Smale [7] introduced a model for computations over the real numbers (and other mathematical structures as well) which is now usually called a BSS machine. It is essentially a random access machine, with the important difference that real numbers are treated as basic entities and that arithmetic operations on the reals are performed in a single step, independently of the magnitude or complexity of the numbers involved. Many basic concepts and fundamental results of computability and complexity theory reappear in the BSS model: the existence of universal machines, the classes $\mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$ (real analogues of P and NP) and the existence of $\mathrm{NP}_{\mathbb{R}}$-complete problems. An example of an $\mathrm{NP}_{\mathbb{R}}$-complete problem is the question whether a given multivariate polynomial of degree four has a real root.

In finite model theory there exist numerous results relating computational complexity with logical definability on finite structures. The subarea investigating such questions is sometimes called *descriptive complexity theory.* The question arises whether similar results can be obtained for complexity over the reals. The main problem for characterising complexity over $\mathbb{R}$ in a model-theoretic setting is to define the right class of structures that permit a clear separation between the finite, discrete aspects of the problems and computations (like indices of tuples, time, indices of registers, the finite control of the machines) on one side and the arithmetic of real numbers on the other side.

It has been shown by Grädel and Meer [16] that this can be achieved by $\mathbb{R}$-structures, a special case of metafinite models, with the ordered field of reals as secondary part. $\mathbb{R}$-structures admit a number of results relating expressibility and complexity that parallel those of descriptive complexity theory in the classical case. In particular, Grädel and Meer established analogues to Fagin's logical characterization of NP in terms generalized spectra [15], and to the Immerman-Vardi Theorem, that fixed point logic captures polynomial time on ordered structures [28, 48]. We will explain some of these results in Sect. 4.

1.4 Related approaches

In database theory there have been a number of proposals for going beyond the strict finiteness condition and incorporating infinite data. In part this was motivated by new areas of application, such as geographical databases, that involve spatial data. We mention a few (by no means all) of the relevant papers.

The study of infinite *recursive structures* has a long tradition in mathematical logic, by the work of Malcev, Nerode, Rabin, Vaught (the order is alphabetical) and their scientific descendents. Recently there have been some papers on recursive structures that study questions related to finite model theory. Hirst and Harel [24] investigated *recursive databases*, given by a finite set of recursive relations over the natural numbers. They studied the notion of a computable query in this context and exhibited complete languages for two specific classes of recursive databases. On the class of all recursive databases, quantifier-free first-order logic suffices to define all computable queries, whereas a variant of QL – the complete language from [10] for the classical relational model — is complete on highly symmetric recursive databases. In another paper Hirst and Harel studied finite model theory issues, such as 0-1 laws and descriptive complexity, in the context of recursive structures [25]. This work is related to ours by the motivation to extend the questions and methods of finite model theory to classes of infinite structures. However, metafinite model theory is radically different from recursive model theory.

Kanellakis, Kuper and Revesz [35] considered databases that are given by semi-algebraic constraints over the real (or rational) numbers. This model can handle spatial data and geometric queries in a very nice and convincing way. Classical relational query languages can be extended with mathematical theories that admit quantifier elimination, such as the theory of real closed fields, to provide a generalized notion of query language, called constraint query languages. Complexity issues of such query languages addressed in [35], and it has been shown that although the decision problem of the underlying mathematical theory may have exponential complexity, the resulting constraint query languages admit efficient evaluation algorithms. In this context we also refer to [19] for some model theoretic results on finitely representable databases.

Kabanza, Stevenne and Wolper [33] present an extension of the relational database model for reasoning abount infinite temporal data. In this model, time is represented by a second sort over the integers and generalized relations are defined by linear constraints, i.e. in Presburger arithmetic. It is proved that first-order queries over such databases can be evaluated in polynomial time.

A proposal that is by far the closest to our approach appears in the penultimate section of the seminal paper of Chandra and Harel [10], the same paper that also laid much of the foundation for the theory of computable queries in the classical, relational model. In that section, Chandra and Harel define the notion of an *extended database*. For a finite domain D and a countable infinite domain F, an extended database is a finite collection of finite mixed relations of the form $R \subseteq D^k \times F^\ell$ and functions of the form $w : D^k \rightarrow F$. Moreover F is "intended to include interpreted features such as numbers, strings (if needed), etc.". In our terminology, an extended database is a metafinite structure with mixed relations. Chandra and Harel define the notion of an extended database query and show that their language QL can be generalized to a complete query language EQL that expresses precisely the extended computable queries. The internal structure of the secondary part F is not really used, except for the assumption that F is effectively enumerable.

As far as we know, this proposal of Chandra and Harel has not been further pursued in database theory, in sharp contrast to the ideas developed in the rest of the paper [10].

2 Metafinite structures

2.1 Basic definitions

In the following, German letters $\mathfrak{A}, \mathfrak{B}, \ldots, \mathfrak{R}, \ldots$, stand for finite or infinite structures; their universes are denoted by corresponding Latin letter $A, B, \ldots, R, \ldots$.

There are many variations of metafinite structures. We define here three basic notions:

- Simple metafinite structures.
- Metafinite structures with multiset operations.
- Metafinite algebras.

Metafinite structures with multiset operations are the most general of these notions, and we will refer to them just as metafinite structures. However, to simplify the exposition we start with the simple variant.

Definition 2.1 A *simple metafinite structure* is a triple $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ consisting of

(i) a finite structure $\mathfrak{A}$, called the primary part of $\mathfrak{D}$;
(ii) a finite or infinite structure $\mathfrak{R}$, called the secondary (or numerical) part of $\mathfrak{D}$.[3] We always assume that $\mathfrak{R}$ contains two distinguished elements 0 and 1 (or TRUE and FALSE);

[3] We denote the numerical part by $\mathfrak{R}$ for "Ar-ithmetic".

(iii) a finite set W of functions $w : A^k \rightarrow R$;

The *vocabulary* of $\mathfrak{D}$ is the triple $\Upsilon(\mathfrak{D}) = (\Upsilon_a, \Upsilon_r, \Upsilon_w)$ where each component of $\Upsilon(\mathfrak{D})$ is the set of relation or function symbols in the corresponding component of $\mathfrak{D}$. (We always consider constants as functions of arity 0.) The two distinguished elements 0,1 of $\mathfrak{R}$ are named by constants of Υ_r.

In finite model theory, we are mostly interested in definability questions concerning *classes* of finite structures. Contrary to classical model theory, a single finite structure often is of lesser interest; for instance, it can be characterized up to isomorphism in first-order logic. Here our main interest are definability questions concerning *classes of metafinite structures with fixed secondary part.* We write $M_\Upsilon[\mathfrak{R}]$ for the class of metafinite structures of vocabulary Υ with secondary part $\mathfrak{R}$ and $\mathrm{Fin}(\Upsilon_a)$ for the class of finite structures with vocabulary Υ_a.

Metafinite structures with multiset operations. Multisets generalize sets in the sense that they allow multiple occurrences of elements. For instance, a function $f : A \rightarrow R$, defines a multiset $\mathrm{mult}(f) = \{\!\{f(a) : a \in A\}\!\}$ over R (the notation $\{\!\{\ldots\}\!\}$ indicates that we allow multiple occurrences of elements). A multiset M over R can also be described by a function $m : R \rightarrow \mathbb{N}$ where $m(r)$ is the multiplicity of r in M. For any set R, let $\mathrm{fm}(R)$ denote the class of all finite multisets over R.

In some of the metafinite structures that we will consider, the secondary part $\mathfrak{R}$ is not just a (first-order) structure in the usual sense, but comes together with a collection of *multiset operations*, i.e. operations $\Gamma : \mathrm{fm}(R) \rightarrow R$, mapping multisets to elements of R. Natural examples on, say, the real numbers are addition, multiplication, counting, mean, maximum, minimum.

Definition 2.2 A *structure with multiset operations* is a pair $\mathfrak{R} = (\mathfrak{R}_0, Op)$ where $\mathfrak{R}_0$ is a first-order structure and Op is a set of operations $\Gamma : \mathrm{fm}(R) \rightarrow R$ (where R is the universe of $\mathfrak{R}_0$). The vocabulary Υ_r of $\mathfrak{R}$ consists of the vocabulary of $\mathfrak{R}_0$ together with the names of the operations in Op. A *metafinite structure with multiset operations* is a triple $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ as in Definition 2.1 with the difference, that $\mathfrak{R}$ is a structure with multiset operations.

Let us give some motivation for this definition. The logics that we will consider contain formulae and terms. Terms may take values in both parts of a metafinite structures. While the rôle of terms over the primary part is rather limited, the terms taking values in the secondary part are called *weight terms* and are of crucial importance here.

A weight term $F(x_1, \ldots, x_k)$ defines, on a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$, a function $F^{\mathfrak{D}} : A^k \rightarrow R$. The collection of values assumed by $F^{\mathfrak{D}}$ forms a *finite multiset*

$$\mathrm{mult}(F^{\mathfrak{D}}) = \{\!\{F^{\mathfrak{D}}(\bar{a}) : \bar{a} \in A^k\}\!\}.$$

We want to have in our languages the expressive means to apply to weight terms natural operations like, say, summation to build the new weight $\sum_{\bar{a}} F^{\mathfrak{D}}(\bar{a})$. Algebraically, this means that we want to have operations mapping multisets over R to elements of R. These multiset operation will allow us to build weight terms from formulae.

Remark. We consider metafinite structures with multiset operations as the default, and will usually refer to them just as metafinite structures.

Metafinite algebras. In principle we can always reduce the primary part of a metafinite structure $\mathfrak{D}$ to a naked set A by pushing all the data into the functions in W. Indeed, we can first replace every function $f : A^k \to A$ by a $(k+1)$-ary relation, and then encode every predicate $Q \subseteq A^k$ by its characteristic function $\chi_Q : A^k \to \{0,1\} \subseteq R$.

Definition 2.3 A *metafinite algebra* is a metafinite structure (with or without multiset operations) whose primary part is a plain set, i.e. $\Upsilon_a = \varnothing$. The elimination of Υ_a-symbols as just described, associates with every metafinite structure $\mathfrak{D}$ a metafinite algebra $\mathfrak{D}^a$, called the *algebraic form* of $\mathfrak{D}$.

As we will explain later, the passage to metafinite algebras permits a lean presentation of a logic as pure calculus of terms. In many cases, this is convenient, in others it is not.

Other variations. There exist several other conceivable variations of metafinite structures that are worth exploring. For instance, instead of allowing only functions from the primary to the secondary part, we may admit *mixed relations* $P \subseteq A^k \times R^m$ or *mixed functions* $f : A^k \times R^m \to R$. Mixed relations may be particularly interesting for database applications; however, to allow for a finite presentation of the databases some restrictions on the admissible relations have to be imposed. A natural restriction is that mixed relations be finite and that mixed functions map all but finitely many elements to 0. But there are other possibilities of finite presentations, e.g., that the relations be recursive [24] or given by semi-algebraic constraints [35, 19].

We won't consider metafinite structures with mixed relations in this paper. However, the design and investigation of query languages for metafinite databases of this kind is one of the promising directions for future research.

Another important variation, in particular for databases, are metafinite structures where the secondary part has *several infinite sorts*, e.g. one for the natural numbers, one for strings, one for real numbers, and so on. While this extension poses no principal difficulty, it often requires heavier notation and we won't consider such structures in this paper.

2.2 Arithmetical structures and $\mathbb{R}$-structures

Of particular interest to us are metafinite structures, whose secondary part is a structure $\mathfrak{N}$ over the natural numbers (or the integers) such that the following hold:

- As a minimum, $\mathfrak{N}$ has the constants 0,1, the functions $+, \cdot$, the ordering relation $<$ and the multiset operations $\max, \min, \sum, \prod$.
- All functions, relations and multiset operations of $\mathfrak{N}$ can be evaluated in polynomial time.

Let us make the second point more precise:

Definition 2.4 Let $\mathfrak{N}_p$ be the structure with the universe $\mathbb{N}$, with all polynomial-time computable functions $f : \mathbb{N}^k \rightarrow \mathbb{N}$ (for all finite arities k) and with all relations $R \subseteq \mathbb{N}^k$ (of arbitrary finite arity k) whose characteristic functions are polynomial-time computable. To define the class of PTIME computable operations on fm($\mathbb{N}$), we have to be a little more careful: we assume that multisets $M \in$ fm($\mathbb{N}$) are represented by listing all elements, repeatedly if they occur more than once. Thus, if $\text{mult}_M(n)$ is the multiplicity of n in M, the *cost* of M is $\|M\| := \sum_n \text{mult}_M(n) \log n$. Now, OpP($\mathbb{N}$) denotes the set of all operations Γ : fm($\mathbb{N}$) $\rightarrow \mathbb{N}$ that are computable in polynomial time (with respect to this representation). *Polynomial-time arithmetic*, denoted PTA, is the pair $(\mathfrak{N}_p, \text{OpP})$.

A *PTA-structure* is a metafinite structure whose secondary part is PTA.

On the other hand, we have as a 'minimal' variant for the secondary part the structure $\mathfrak{N}_0 = (\mathbb{N}, 0, 1, +, \cdot, <, \max, \min, \sum, \prod)$.

Definition 2.5 An *arithmetical structure* is a metafinite structure with secondary part $\mathfrak{N}$ such that $\mathfrak{N}$ is an expansion of $\mathfrak{N}_0$ and a reduct of PTA. A *simple arithmetical structure* is obtained from an arithmetical structure by omitting the multiset operations.

Another interesting class are $\mathbb{R}$-structures, used by Grädel and Meer [16] for developing a descriptive complexity theory over the real numbers.

Definition 2.6 An $\mathbb{R}$-structure is a simple metafinite structure with secondary part

$$\mathfrak{R} = (\mathbb{R}, +, -, \cdot, /, \leq, (c_r)_{r \in \mathbb{R}}).$$

It is convenient to include subtraction and division as primitive operations and assume that every element $r \in \mathbb{R}$ is named by a constant c_r so that every rational function $g : \mathbb{R}^k \rightarrow \mathbb{R}$ can be written as a term (without quantifiers).

In [16] a slightly different presentation has been used including also the sign function

$$\text{sgn}(x) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

as a basic function. Clearly, this function is efficiently computable, but is not a rational function. We don't need this function here, because we have chosen to include in our logics a characteristic function rule (see Definition 3.1) from which the sign function is easily definable.

2.3 Global functions, numerical invariants and their complexity

Let $\mathcal{K}$ be a class of metafinite structures with secondary part $\mathfrak{R}$.

Definition 2.7 A *global function* on $\mathcal{K}$ of arity k is a function F that assigns to every structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W) \in \mathcal{K}$ a (local) function $F^{\mathfrak{D}} : A^k \rightarrow R$ such that isomorphisms between structures are preserved: for every isomorphism $h : \mathfrak{D} \rightarrow \mathfrak{D}'$ we have that for all $a_1, \ldots, a_k \in A$

$$hF^{\mathfrak{D}}(a_1, \ldots, a_k) = F^{\mathfrak{D}'}(ha_1, \ldots, ha_k).$$

In most cases, $\mathfrak{R}$ will be a "numerical" structure (e.g. the natural numbers with arithmetical operations, or the field of rational, real or complex numbers) which is rigid, thus the restriction to the secondary part of any isomorphism between structures of $\mathcal{K}$ is the identity on $\mathfrak{R}$. In this case, we call a nullary global function — assigning to each isomorphism class of structures a numerical value — a *numerical invariant.*

There are many interesting examples of numerical invariants both in the case of structures without weights and in the case of structures with weights: the order of the automorphism group of the structure, in graph theory the usual graph parameters like the chromatic number, clique number or genus, and also the cost of an optimal solution for an optimization problem, like the length of a shortest TSP tour. Examples of global numerical functions of positive arity are the distance between vertices x, y of a given graph, the order of an element x of a given group (i.e. the cardinality of the cyclic subgroup generated by x), etc.

Our notion of global functions generalizes the notions of global functions and global relations (or relational queries) in finite model theory and databases. Thus questions concerning computability, complexity and expressibility of relational queries on finite structures can be viewed as special cases of the corresponding questions on global functions on classes of metafinite structures.

Complexity of global functions. A notion of complexity for global functions may very much depend on the computational model under consideration, and on the *cost* associated with the elements of the secondary part. For instance, if the secondary part consists of natural numbers or binary strings, then we have a natural notion of cost given by the number of bits. On the other side, if we study complexity over real numbers with respect to the Blum-Shub-Smale model, then we treat every element of $\mathbb{R}$ as a basic entity of cost one.

To obtain a flexible and general notion of complexity of global functions it is therefore convenient to associate with the secondary part $\mathfrak{R}$ a *cost function*

$$\| \, \| : R \rightarrow \mathbb{N}.$$

The cost of a weight function $w : A^k \rightarrow R$ is then defined as $\|w\| := \sum_{\bar{a} \in A^k} \|w(\bar{a})\|$. The cost of a metafinite algebra $\mathfrak{D} = (A, \mathfrak{R}, W)$ is $\|\mathfrak{D}\| = \sum_{w \in W} \|w\|$ and the cost of a metafinite structure can be defined as the cost of the associated metafinite algebra. Note that this cost is always finite, and that the secondary part — which is assumed to be fixed — is given for free.

Proviso. For arithmetical structures, we let $||n|| = 1 + \lfloor \log n \rfloor$, i.e. the length of the binary representation of n (with the convention that $\log 0 = 0$). For $\mathbb{R}$-structures, our default is that $||r|| = 1$ for all $r \in \mathbb{R}$, which reflects the use of $\mathbb{R}$-structures for capturing complexity classes with respect to the Blum-Shub-Smale model.

For a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ we write $|\mathfrak{D}|$ for the cardinality of the primary part $\mathfrak{A}$ and let

$$\max\mathfrak{D} := \max_{w \in W} \max_{\bar{a}} ||w(\bar{a})||$$

be the cost of the maximal weight. Then $||\mathfrak{D}|| \leq p(|\mathfrak{D}|, \max\mathfrak{D})$ for some polynomial $p(n, m)$ that depends only on the vocabulary of $\mathfrak{D}$. Since most of the popular complexity classes are invariant under polynomial increase of the relevant input parameters, it therefore makes sense to measure the complexity of a computation on a structure $\mathfrak{D}$ in terms of $|\mathfrak{D}|$ and $\max\mathfrak{D}$.

For instance, an algorithm M on a class $\mathcal{C}$ of metafinite structures runs in polynomial-time (respectively, logarithmic space) if, on every input $\mathfrak{D} \in \mathcal{K}$, the computation of M terminates in at most $q(|\mathfrak{D}|, \max\mathfrak{D})$ steps, for some polynomial q, (respectively, uses at most $O(\log|\mathfrak{D}| + \log\max\mathfrak{D})$ of work space).

More generally, we can define the following notion of complexity

Definition 2.8 Let $\mathcal{K}$ be a class of metafinite structures with secondary part $\mathfrak{R}$, and $|| \; || : R \to \mathbb{N}$ a cost function. Let $\mathcal{M}$ be a machine model, suitable for evaluating global functions on $\mathcal{C}$. A *resource measure* for $\mathcal{M}$ is a function T associating with every $\mathcal{M}$-algorithm M and every input x a number $T_M(x) \in \mathbb{N} \cup \{\infty\}$. We say that M evaluates the global F on $\mathcal{C}$ with resource bound $t(n, m)$ if, given any structure $\mathfrak{D} \in \mathcal{K}$, and any tuple $\bar{a}$ of appropriate length for F, M computes $F^{\mathfrak{D}}(\bar{a})$ in such a way that $T_M(\mathfrak{D}, \bar{a}) \leq t(|\mathfrak{D}|, \max\mathfrak{D})$.

3 Logics of metafinite structures.

Fix any logic L (on finite structures), e.g. first-order logic, fixed point logic or the infinitary logic $L^{\omega}_{\infty\omega}$. There are several ways to extend L to a logic of metafinite structures.

3.1 Simple languages.

The first such extension, let us call it L^* for the time being, is suitable for reasoning on simple metafinite structures. It is given by the following definition.

Definition 3.1 Let $\Upsilon = (\Upsilon_a, \Upsilon_r, \Upsilon_w)$ be a vocabulary of simple metafinite structures (i.e. Υ_r does not contain multiset operations). Fix a countable set $V = \{x_0, x_1, \ldots, \}$ of variables. *These variables range only over the primary part; we don't use variables taking values in the secondary part.*

The language of $L^*(\Upsilon)$ contains the following expressions:

- Terms over the primary part, denoted by $t_1, t_2, \ldots$ which are called *point terms*. On a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$, a point term $t(x_1, \ldots, x_k)$ defines a function $t^{\mathfrak{D}} : A^k \to A$.
- Terms over the secondary part, which are called *weight terms* and are denoted by $F, G, H, \ldots$. On $\mathfrak{D}$, a weight term $F(x_1, \ldots, x_k)$ defines a weight function $F^{\mathfrak{D}} : A^k \to R$.
- *Formulae.* On $\mathfrak{D}$, a formula $\varphi(x_1, \ldots, x_k)$ defines a predicate $\varphi^{\mathfrak{D}} = \{\bar{a} : \mathfrak{D} \models \varphi(\bar{a})\}$.

The terms, weights and formulae of $L^*(\Upsilon)$ are defined inductively by the following rules:

(i) the set of point terms is the closure of the set of V of variables under applications of function symbols of Υ_a.

(ii) If $t_1, \ldots, t_k$ are point terms and w is a k-ary function symbol of Υ_w then $w(t_1, \ldots, t_k)$ is a weight term.

(iii) If $F_1, \ldots, F_k$ are weight terms and g is a k-ary function symbol of Υ_r then $g(F_1, \ldots, F_k)$ is a weight term. In particular, all all *closed* terms (in the usual sense) over Υ_r are weight terms of $L^*(\Upsilon)$.

(iv) Atomic formulae are either equalities of point terms, or equalities of weight terms, or expressions $P(t_1, \ldots, t_k)$ or $Q(F_1, \ldots, F_k)$ where P and Q are k-ary predicate symbols in Υ_a and Υ_r, respectively.

(v) The set of formulae of L^* is closed under all rules of L for building formulae. However, note that all variables appearing in these formulae range over the primary part only.

(vi) The characteristic function rule: If φ is a formula of L^*, then $\chi[\varphi]$ is a weight term of L^*, with the same free variables as φ and the semantic

$$\chi[\varphi]^{\mathfrak{D}}(\bar{a}) = \begin{cases} 1 & \text{if } \mathfrak{D} \models \varphi(\bar{a}) \\ 0 & \text{otherwise.} \end{cases}$$

The *basic* terms are the point terms and the weight terms that can be built using only the rules *(i)* - *(iv)* and *(vi)*. Note that the set of basic terms depends only on Υ, not on L.

3.2 Logics with multiset operations

We now turn to logics that make use of multiset operations. As described in Sect. 2.1, multiset operations can be used to define terms from formulae. They play a similar rôle as quantifiers do, in fact quantifiers can be viewed as a special form of multiset operations.

In the case where Υ_w contains multiset operations, we add to the inductive definition of $L^*(\Upsilon)$ the following **multiset operation rule**:

Syntax: Let $\bar{x}$ and $\bar{y}$ be tuples of variables, $F(\bar{x}, \bar{y})$ be a weight terms of vocabulary Υ and φ be a formula of vocabulary Υ. Then, for every multiset operation Γ of Υ_r, the expression

$$G(\bar{y}) := \Gamma_{\bar{x}}(F(\bar{x}, \bar{y}) : \varphi)$$

is a weight term of vocabulary Υ, with free variables $\bar{y}$.

Semantic: The interpretation of $G(\bar{y})$ on an Υ-structure $\mathfrak{D}$ with valuation $\bar{b}$ for $\bar{y}$ is

$$G^{\mathfrak{D}}(\bar{b}) := \Gamma\{\!\{F^{\mathfrak{D}}(\bar{a}, \bar{b}) : \mathfrak{D} \models \varphi(\bar{a}, \bar{b})\}\!\}.$$

To enhance readability, we will sometimes omit the free variables and use the abbreviated notation $\Gamma_{\bar{x}}(F : \varphi)$. Furthermore, we will omit φ in case it is a tautology and write $\Gamma_{\bar{x}}F(\bar{x}, \bar{y})$.

There are some important multiset operations that are invariant under adding arbitrary occurrences of 0 to the multiset: $\Gamma(S) = \Gamma(S \cup \{\!\{0, 0, \ldots, 0\}\!\})$ for all $S \in \text{fm}(R)$. For instance, this is the case for $\sum$ and max. In this case, we can use $(\Gamma_{\bar{x}}\ F \cdot \chi[\varphi])$ rather than $\Gamma_{\bar{x}}(F : \varphi)$.

Example 3.2 [Binary representations] Consider arithmetical structures with primary part of the form $\mathfrak{A} = (\{0, \ldots, n-1\}, <, P)$ where P is a unary relation. P is interpreted as a bit sequence $u_0 \cdots u_{n-1}$ representing the natural number $\sum_{i=0}^{n-1} u_i 2^i$ (where $u_i = 1$ iff $\mathfrak{A} \models P(i)$). The number represented by P is definable by the term

$$\sum_x (\chi[Px] \prod_y (2 : y < x)).$$

Example 3.3 [Counting elements] On arithmetical structures, we can count in FO*. For any formula $\varphi(\bar{x})$ there is a weight term $\#_{\bar{x}}[\varphi(\bar{x})]$ counting the number of tuples $\bar{a}$ such that $\varphi(\bar{a})$ is true. Indeed,let

$$\#_{\bar{x}}[\varphi(\bar{x})] := \sum_{\bar{x}} \chi[\varphi].$$

Example 3.4 [Counting equivalence classes] Let $\mathfrak{D}$ be an arithmetical structure and $\varphi(x, y)$ be a binary formula, defining an equivalence relation $\sim_\varphi$ on A. If we have division as a basic function in $\mathfrak{N}$, then the index of $\sim_\varphi$, denoted $\#[A/\varphi]$, is definable in FO* in the following way.

By the previous example, $F(x) = \#_y[\varphi(x, y)]$ is a weight term of FO*. The index of $\sim_\varphi$ can be written as a sum of rational numbers: $\#[A/\varphi] = \sum_x (F(x))^{-1}$. To do everything over $\mathbb{N}$, let $G = \prod_x F(x)$; thus the weight $G/F(x)$ is also FO*-definable and we get

$$\#[A/\varphi] = \Big(\sum_x G/F(x)\Big)/G.$$

Multiset operations play an important rôle in metafinite model theory. They partially compensate for the limited access to the secondary part and greatly enhance the expressive power of the logics that we consider. We also believe that they provide the right logical formalism for the *aggregate operators* used in databases (see Chapter 7.3: "Confronting the Real World" in [1]).

3.3 An excursion: Reliability of queries

We present here a more elaborate example for the use of multiset operations that addresses the issue of fault tolerance of relational queries as mentioned in Sect. 1.3.

Definition 3.5 An *unreliable database* is a pair $(\mathfrak{A}, \mu)$ where $\mathfrak{A}$ is a finite structure and μ a probability function on the of atomic statements $R(\bar{a})$ about $\mathfrak{A}$.

Think about $\mathfrak{A}$ as the observed database. For every first-order statement $\varphi(\bar{a})$ about $\mathfrak{A}$, let Wrong$(\varphi(\bar{a}))$ be the event that the truth-value of $\varphi(\bar{a})$ in $\mathfrak{A}$ differs from the truth-value of $\varphi(\bar{a})$ in the actual database. $\mu(R(\bar{a}))$ is the probability of the event Wrong$(R(\bar{a}))$. It is supposed that the events Wrong$(R(\bar{a}))$ are independent.

Let $\mathfrak{B}$ be a database of the same vocabulary as $\mathfrak{A}$ and with the same universe as $\mathfrak{A}$. Let $D(\mathfrak{B})$ be the collection of atomic statements $R(\bar{a})$ that are true in $\mathfrak{B}$. The probability that $\mathfrak{B}$ is the actual database is

$$\nu(\mathfrak{B}) := \prod_{\varphi \in |D(\mathfrak{A}) - D(\mathfrak{B})|} \mu(\varphi) \prod_{\varphi \in D(\mathfrak{A}) \Leftrightarrow \varphi \in D(\mathfrak{B})} (1 - \mu(\varphi))$$

where $|D(\mathfrak{A}) - D(\mathfrak{B})|$ is the symmetric difference.

Given a relational query $\psi(\bar{x})$ of arity k, let $\psi^{\mathfrak{A}} = \{\bar{a} \in A^k : \mathfrak{A} \models \psi(\bar{a})\}$. The Hamming distance between $\psi^{\mathfrak{A}}$ and $\psi^{\mathfrak{B}}$ is the cardinality of the symmetric difference $|\psi^{\mathfrak{A}} - \psi^{\mathfrak{B}}|$.

Definition 3.6 With every tuple $\bar{a} \in A^k$, we associate the random variable $P_\psi(\bar{a})$ that assigns to a database $\mathfrak{B}$ the probability that $\psi(\bar{a})$ distinguishes between $\mathfrak{B}$ and the observed database $\mathfrak{A}$. Summing up the probabilities $P_\psi(\bar{a})$ over all tuples $\bar{a} \in A^k$ gives the expectation $E(H_\psi)$ of the Hamming distance between $\psi^{\mathfrak{A}}$ and $\psi^{\mathfrak{B}}$. The number $F_\psi := 1 - [E(H_\psi)/n^k]$ is the *fault-tolerance* of ψ.

For simplicity, we restrict attention to conjunctive queries, that is to queries of the form

$$\exists x_1 \ldots \exists x_k [\varphi_1 \wedge \cdots \wedge \varphi_l]$$

where each φ_i is an atomic or negated atomic formula. It is supposed that the atomic parts of formulas φ_i and φ_j are different if $i \neq j$.

Unreliable databases can be modelled by metafinite structures where the secondary part $\mathfrak{R}$ is the field of reals with the multiset operations $\sum, \prod$. We check that the expected Hamming distance and the fault-tolerance of the conjunctive queries are *first-order definable* numerical invariants.

Let $(\mathfrak{A}, \mu)$ be an unreliable database of relational vocabulary Υ_a. View μ as a tuple of probability functions μ_R where R is a proper predicate (not the equality sign) in Υ_a (and μ_R is the restriction of μ to atoms of the form $R(\bar{a})$. With an unreliable database$(\mathfrak{A}, \mu)$, we associate the metafinite structure $(\mathfrak{A}, \mathfrak{R}, \{\mu_R : R \in \Upsilon_a\})$.

Proposition 3.7 *Let $\psi(\bar{x})$ be a subformula of a conjunctive query as above. Then*

- *(i) $P_\psi(\bar{x})$ is a first-order definable global weight function;*
- *(ii) The expected Hamming distance and the fault-tolerance of ψ are first-order definable numerical invariants.*

Proof. Since $E(H_\psi) = \sum_{\bar{x}} \mu_\psi(\bar{x})$, and $F_\psi = 1 - E(H_\psi)/n^k$ the second claim follows from the first.

The first claim is proved by induction on $\psi(\bar{x})$. If ψ is an atom $R(\bar{x})$ then clearly $P_\psi(\bar{x}) = \mu_R(\bar{x})$ which is also atomic. Obviously $P_{\neg\psi}(\bar{x}) = P_\psi(\bar{x})$.

Let $N(x_1, \dots, x_k)$ be the assertion that $x_i \neq x_j$ if $i < j$, and let $N(\psi(\bar{x})) = N(\bar{x})$. It suffices to find a weight term that expresses P_ψ only in the case when $N(\psi)$ hold. Indeed, let $\alpha_1, \dots, \alpha_m$ be all different consistent assertions about the equality relation on the components of $\bar{x}$. P_ψ is the sum of probabilities $P_{\alpha_i \wedge \psi}$, and each $\alpha_i \wedge \psi$ is equivalent to a formula of the form $N(\bar{y}) \wedge \varphi(\bar{y})$.

Now suppose that $\psi(\bar{x})$ is a conjunction $\psi_1(\bar{x}) \wedge \psi_2(\bar{x})$ where $\bar{x} = (x_1, \dots, x_k)$ and restrict attention to the case when $N(\bar{x})$ holds. The events Wrong($\psi_i(\bar{a})$) are independent.

In case $\mathfrak{A} \models \psi(\bar{a})$, Wrong($\psi(\bar{a})$) = Wrong($\psi_1(\bar{a})$) ∪ Wrong($\psi_2(\bar{a})$). In case $\mathfrak{A} \models (\neg\psi_i(\bar{a}) \wedge \psi_{3-i}(\bar{a}))$, Wrong($\psi(\bar{a})$) = Wrong($\psi_i(\bar{a})$). Finally, in case $\mathfrak{A} \models (\neg\psi_1(\bar{a}) \wedge \neg\varphi_2(\bar{a})$, Wrong($\psi(\bar{a})$ = Wrong($\psi_1(\bar{a})$) ∩ Wrong($\psi_2(\bar{a})$).

Recall that the characteristic function $\chi[\psi]$ of any first-order formula is a first-order weight term. We have:

$$
\begin{aligned}
P_\psi = {} & \chi[\psi][1 - (1 - P_{\psi_1})(1 - P_{\psi_2})] \\
& + \chi[\neg\psi_1 \wedge \psi_2] P_{\psi_1} \\
& + \chi[\psi_1 \wedge \neg\psi_2] P_{\psi_2} \\
& + \chi[\neg\psi \wedge \neg\psi_2] P_{\psi_1} P_{\psi_2}
\end{aligned}
$$

Finally, suppose that $\psi(\bar{x}) = (\exists y)(\varphi(\bar{x}, y))$ and restrict attention to the case that $N(\bar{x})$ holds. It suffices to find a weight term that expresses the probability $p = P_{\psi(\bar{x})}$ in the case that that $N(\bar{x}, y)$ holds. Indeed, fix an instantiation $\bar{a}$ of $\bar{x}$ such that the components $a_1, \dots, a_k$ of $\bar{a}$ are distinct. $P_{\psi(\bar{a})}$ is the sum of p and the probabilities $P_{\varphi(\bar{a}, a_i)}$. Now use the induction hypothesis.

Restrict attention to the case that $N(\bar{x}, y)$ holds. Without loss of generality, every conjunct of φ contains y. Indeed, let $\alpha(\bar{x}, y)$ be the quantifier-free part of φ, and $\beta(\bar{x})$ be the conjunction of those conjuncts of α that do not contain y. Clearly, $\psi(\bar{x})$ is equivalent to $\beta(\bar{x}) \wedge (\exists y)\alpha(\bar{x}, y)$. Furthermore, the events Wrong($\beta(\bar{a})$ and Wrong($(\exists y)\alpha(\bar{x}, y)$) are independent and thus the conjunction can be treated as above. Now use the induction hypothesis.

Again, fix an instantiation $\bar{a}$ of $\bar{x}$ such that the components $a_1, \dots, a_k$ of $\bar{a}$ are distinct. Since every conjunct of φ contains y, statements $\varphi(\bar{a}, b)$, where b ranges over $A - \{a_1, \dots, a_k\}$, are pairwise independent.

In case $\mathfrak{A} \models \psi(\bar{a})$, $\text{Wrong}(\psi) = \bigcap_{b \in B} \text{Wrong}(\varphi(\bar{a}, b)$ where B is the collection $b \in A - \{a_1, \ldots, a_k\}$ such that $\mathfrak{A} \models \varphi(a, b)$. In case $\mathfrak{A} \models \neg\psi(\bar{a})$, $\text{Wrong}(\psi(\bar{a})) = \bigcup_b \text{Wrong}(\varphi(\bar{a}, b))$. We have

$$\begin{aligned} P_\psi = \chi[\psi] \cdot \prod (P_\varphi : \varphi(\bar{x}, y) \wedge N(\bar{x}, y)) + \\ \chi[\neg\psi] \cdot [1 - \prod (1 - P_\varphi : N(\bar{x}, y))]. \end{aligned}$$

□

3.4 Pure term calculi

We now explain how, for metafinite algebras, logics can be presented as pure calculi of weight terms. We first assume, for simplicity, that the secondary part $\mathfrak{R}$ is an algebra, i.e. the vocabulary Υ_r contains no relation symbols. Thus we deal with vocabularies $\Upsilon = (\Upsilon_r, \Upsilon_w)$ consisting of two sets of function symbols.

Definition 3.8 The calculus $\text{FOT}(\Upsilon)$ of *first-order terms* of vocabulary Υ, together with the notion of the *rank* of a term, is defined inductively as follows:

(i) If $x_1, \ldots, x_k$ are variables, and w is a k-ary weight function in Υ_w, then $w(x_1, \ldots, x_m)$ is a term of rank 0 in $\text{FOT}(\Upsilon)$.
(ii) If $F_1, \ldots, F_m \in \text{FOT}(\Upsilon)$ and $g \in \Upsilon_r$ is a m-ary function symbol, then $g(F_1, \ldots, F_m)$ is a term of $\text{FOT}(\Upsilon)$, whose rank is the maximum of the ranks of $F_1, \ldots, F_m$.
(iii) If F and G belong to $\text{FOT}(\Upsilon)$, then so does $\chi[F = G]$. The rank of $\chi[F = G]$ is the maximum of the ranks of F and G.
(iv) If F and G belong to $\text{FOT}(\Upsilon)$, $\bar{y}$ is an ℓ-tuple of variables and Γ a multiset operation from Υ_w, then $\Gamma_{\bar{y}}(F(\bar{x}, \bar{y}) : G = 1)$ is a term of $\text{FOT}(\Upsilon)$, of rank $\ell + \max\{\text{rk}(F), \text{rk}(G)\}$.

For terms formed with multiset operations, we also use a simplified form $\Gamma_{\bar{y}}\ F(\bar{x}, \bar{y})$ as an abbreviation for $(\Gamma_{\bar{y}}\ (F(\bar{x}, \bar{y}) : 1 = 1)$.

It is clear that for, say, arithmetical structures, or for $\mathbb{R}$-structures with maximization as a multiset operation, first-order logic can be simulated by FOT, in the sense that the characteristic function of every first-order formula is equivalent to a term in FOT. Indeed, this follows by a straightforward induction using the following equalities:

$$\begin{aligned} \chi[\psi \wedge \varphi] &= \chi[\psi]\chi[\varphi] \\ \chi[\neg\psi] &= 1 - \chi[\psi] \\ \chi[\exists x \psi] &= \max_x(\chi[\psi]). \end{aligned}$$

In fact, this holds for all secondary parts as long as we have two definable functions $\wedge$ and $\neg$, interpreted on $\{0, 1\} \subseteq R$ in the usual way, and any multiset operation that distinguishes, say, multisets with occurrences of 1 from those without (or the empty multiset from the nonempty ones).

Remark. The restriction to algebraic structures is not necessary. When we deal with an arbitrary vocabulary $\Upsilon = (\Upsilon_a, \Upsilon_r, \Upsilon_w)$ for metafinite structures, we can still present first-order logic as a pure calculus of weight terms. We just have to replace in clause *(i)* of Definition 3.8 the variables X_i by arbitrary point terms over Υ_a (as in clause *(ii)* of Definition 3.1), and add the rules defining for every predicate Q and already defined terms $F_1, \ldots, F_m$ also the term $\chi[Q(F_1, \ldots, F_m)]$.

3.5 Second-order multiset operations

In several contexts, for instance for dealing with NP-optimization problems or with counting problems in the class #P, it is convenient to have logics with second-order constructs.

Multiset operations can be viewed as a generalization of quantifiers. Therefore, natural variants of second order logics can be defined by applying multiset operations to predicate variables.

Definition 3.9 Suppose we have a logic L in the usual sense (say, second-order logic or it existential fragment Σ^1_1), then L^{**} is the smallest logic closed under the rules of L^* together with the following rule.

Multiset operation rule (second order):

Syntax: Let $\Upsilon = (\Upsilon_a, \Upsilon_r, \Upsilon_w)$ be a vocabulary and $\Upsilon' = (\Upsilon_a \cup \{\bar{X}\}, \Upsilon_r, \Upsilon_w)$ where $\bar{X}$ is a tuple of relation variables. If F is a weight term and φ a formula of vocabulary Υ' with free variables among $\bar{x}, \bar{y}$, then, for every multiset operation Γ of Υ_r, the expression

$$\Gamma_{\bar{X},\bar{x}}(F : \varphi)$$

is a weight term of vocabulary Υ, with free variables $\bar{y}$.

Semantic: The interpretation of this expression on an Υ-structure $\mathfrak{D}$ with valuation $\bar{b}$ for $\bar{y}$ is

$$\Gamma\{\!\{F^{(\mathfrak{D},\bar{X})}(\bar{a},\bar{b}) : (\bar{X},\bar{a}) \text{ satisfy } (\mathfrak{D},\bar{X}) \models \varphi(\bar{a},\bar{b})\}\!\}.$$

Example 3.10 [The Travelling Salesman Problem] NP-optimization problems like the TSP can be expressed in a very direct way in this framework, since the arithmetic that is necessary to determine the length of a tour and to minimize is separated from the graph.

Let $order(<)$ express that $<$ is a linear ordering, and let $succ(<, x, y)$ be a formula which, for any given linear ordering $<$, says that either y is the successor of x, or x is the maximal and y the minimal element of the ordering. Then the length of the shortest tour of any instance (V, w) of the TSP, where $w : V \times V \rightarrow \mathbb{N}$ is the weight function giving the distances, is defined by the weight

$$opt_{\mathrm{TSP}}(V, w) = \min_{<}\Big(\sum_{x,y}(w(x,y) : succ) : order\Big).$$

A more challenging example: the genus of a graph. The *genus* $\gamma(G)$ of an undirected graph G is the smallest $g \in \mathbb{N}$ such that G can be embedded into the sphere with g handles.

The genus is one of the most important graph parameters. It is hard to compute; the corresponding decision problem — given a graph G and a number k, decide whether $\gamma(G) \leq k$ — is NP-complete.

It is more convenient for us to work with a different, purely combinatorial characterization of the genus.

Definition 3.11 A *rotation system* on a undirected graph $G = (V, E)$ is a ternary predicate $P \in V^3$ which defines for every node a cycle on the edges incident to it. More precisely: if $(x, y, z) \in P$, then $(x, y) \in E$ and $(y, z) \in E$, and for all $y \in V$, the directed graph $H_y = (S_y, C_y)$ with

$$S_y := \{x : (x, y) \in E\}$$
$$C_y := \{(x, z) : (x, y, z) \in P\}$$

is a cycle. A *P-face* is defined by a cycle $x_0, \ldots, x_{r-1}$ in G such that, for all $i < r$, $(x_{i-1}, x_i, x_{i+1}) \in P$ (here, indices are expressed modulo r). The *P-genus* of G, denoted $\gamma(P)$ is defined by Euler's formula

$$n - e + f(P) = 2 - 2\gamma(P)$$

where n is the number of vertices, e the number of edges and $f(P)$ the number of P-faces.

The following result is well-known in graph theory (see e.g. [18])

Proposition 3.12 *The genus of G is the minimal P-genus of G.*

For convenience our logical definition of the genus is based on transitive closure logic. This is a familiar logic in finite model theory which augments first-order logic by the ability to define transitive closures. It admits, for every formula $\varphi(\bar{x}, \bar{y})$ with k-tuples $\bar{x}, \bar{y}$ of free variables, also the formula $[\mathrm{TC}_{\bar{x},\bar{y}}\ \varphi](\bar{a}, \bar{b})$ expressing that $(\bar{a}, \bar{b})$ is contained in the reflexive and transitive closure of the binary relation that φ defines on k-tuples.

It is easy to see that there exists a formula ψ of vocabulary $\{E, P\}$ in transitive closure logic such that for every graph G and every ternary predicate P on G

$(G, P) \models \psi$ if and only if P is a rotation system on G.

The number of P-faces is the number of equivalence classes of directed edges with respect to the reachability relation defined by P. It is not difficult to construct a formula $\alpha(P, Q)$ in transitive closure logic saying that Q is a binary relation containing at most one directed edge on each P-face:

$$\alpha(P,Q) = \forall x \forall y \forall u \forall v((Qxy \wedge Quv \wedge.\\ [\mathrm{TC}_{xy,uv} y = u \wedge P(x,y,v)](xy,uv)) \rightarrow (x,y) = (u,v)).$$

Given that $\psi(P)$ expresses that P is a rotation system, that the weight $\#Q$ is definable in $FO^*(\{Q\})$ and that n and e are obviously definable, we can define the genus of an undirected graph by

$$\gamma = 1 + \frac{1}{2}(e - n - \max_{P,Q}(\#Q : \psi \wedge \alpha)).$$

4 Descriptive complexity

One of the goals of metafinite model theory is the *descriptive complexity theory of problems with weights.* For finite models, the results of Fagin, Immerman, Vardi and others provide logical characterizations of NP, P and also for most of the other important complexity classes, at least on ordered structures. We refer to [21, 29, 30] for surveys on descriptive complexity.

Here we investigate generalizations of these results in the realm of metafinite structures. For simplicity, we focus on arithmetical structures; we also mention $\mathbb{R}$-structures but refer to [16] for proofs. However, the approach can be extended to problems on metafinite structures with arbitrary secondary part. This requires the definition of a suitable machine model and a suitable notion of complexity. We will defer the detailed development to a subsequent paper.

We start with the observation that first-order logic can be evaluated in polynomial-time.

Proposition 4.1 *If the basic functions, relations and multiset operations of $\mathfrak{R}$ can be evaluated in polynomial time (with respect to the given cost function), then the same is true for every first-order definable global function on $M_\Upsilon[\mathfrak{R}]$.*

The proof is a straightforward induction.

4.1 Metafinite spectra

We first consider Fagin's characterization of NP by existential second-order logic [15].

Definition 4.2 [Fagin] A class $\mathcal{K}$ of finite Υ_a-structures is a *generalized spectrum* if there exists a first-order sentence ψ of a vocabulary $\Upsilon_a \cup \{R_1, \ldots, R_m\}$ such that $\mathfrak{A} \in \mathcal{K}$ if and only if there exists an expansion $\mathfrak{B}$ of $\mathfrak{A}$ with $\mathfrak{B} \models \psi$.

Remark. An equivalent definition is that a generalized spectrum is the class of finite models of an existential second-order sentence $\exists R_1 \cdots \exists R_m \psi$. However, as discussed below, there are several possibilities of generalizing second-order logic to metafinite structures, and we don't want to commit ourselves to one particular variant. We will therefore mostly work with (generalizations of) the definition given above.

Informally, Fagin's Theorem states that the generalized spectra are precisely the model classes recognizable in nondeterministic polynomial time. For a precise statement of this result, we have to keep in mind that to serve as an input for a classical computational device like a Turing machine, a finite structure needs to be encoded by a string. At least implicitly, such an encoding requires that an ordered representation of the structure is chosen. The precise form of the encoding is not important, as long as it satisfies some reasonable simple properties. So when we say that a class of structures is in NP we actually mean that the set of encodings of structures in that class is in NP.

Theorem 4.3 (Fagin) *Let $\mathcal{K}$ be a class of finite structures of a fixed finite vocabulary which is closed under isomorphisms. Then $\mathcal{K}$ is in NP if and only if it is a generalized spectrum.*

Does Fagin's Theorem generalize to metafinite structures? To address this problem, we need to make precise two notions:

- The notion of a *metafinite spectrum*, i.e. a generalized spectrum of metafinite structures.
- The notion of nondeterministic polynomial time complexity in the context of metafinite structures.

We start with two notions of metafinite spectra.

Definition 4.4 A class $\mathcal{K} \subseteq M_\Upsilon[\mathfrak{R}]$ is a *metafinite spectrum* if there exists a first-order sentence ψ of a vocabulary $\Upsilon' \supseteq \Upsilon$ such that $\mathfrak{D} \in \mathcal{K}$ if and only if there exists an expansion $\mathfrak{D}' \in M_{\Upsilon'}[\mathfrak{R}]$ of $\mathfrak{D}$ with $\mathfrak{D}' \models \psi$. A *primary metafinite spectrum* is defined in a similar way, except that only the primary part of the structures is expanded. This means that the expanded structures $\mathfrak{D}'$ have the same set of weight functions as $\mathfrak{D}$.

Remark. These two notions of metafinite spectra correspond to two variants of (existential) second-order logic. The more restrictive one allows second-order quantifiers only over primary relations, whereas the general one allows quantification over weight functions as well. Thus, a primary metafinite spectrum is the class of models $\mathfrak{D} \in M_\Upsilon[\mathfrak{R}]$ which are models of an existential second-order sentence of the form $\exists R_1 \cdots \exists R_m \psi$ where $R_1, \ldots, R_m$ are relation variables over the primary part and ψ is first-order (in the sense of Definition 3.1). Since relations over the primary part can be replaced by their characteristic functions, a metafinite spectrum in the more general sense is the class of models of a sentence $\exists F_1 \cdots \exists F_m \psi$ where F_i are function symbols ranging over weight functions.

4.2 Generalizations of Fagin's Theorem

We show, that both notions of metafinite spectra capture (suitable variants of) nondeterministic polynomial-time in certain contexts, but fail to do so in others.

First we consider arithmetical structures where the secondary part is $\mathfrak{N}$, as given by Definition 2.5. We assume that that the cost of natural numbers is given by the length of their binary representations. As described in Sect. 2.3, this gives a natural notion of the complexity of global functions, and in particular of an NP-class of arithmetical structures. So the question is, whether, or under what circumstances NP is captured by the class of metafinite spectra or primary metafinite spectra.

The original proof of Fagin's Theorem generalizes to the case of arithmetical structures with not too large weights.

Definition 4.5 A class $\mathcal{K}$ of metafinite structures has *small weights* if there exists a $k \in \mathbb{N}$ such that $\max\mathfrak{D} \leq |\mathfrak{D}|^k$ for all $\mathfrak{D} \in \mathcal{K}$.

Recall that $\max\mathfrak{D}$ stands for the cost of the largest weight. Thus, a class of arithmetical structures has small weights if the values of the weights are bounded by a function $2^{p(|\mathfrak{D}|)}$ for some polynomial p. We obtain the following first generalization of Fagin's result.

Theorem 4.6 *Let $\mathcal{K} \subseteq M_{\Upsilon}[\mathfrak{N}]$ be a class of arithmetical structures with small weights, which is closed under isomorphisms. The following are equivalent:*

(i) $\mathcal{K}$ is in NP.
(ii) $\mathcal{K}$ is a primary generalized spectrum.

Proof. It is obvious that *(ii)* implies *(i)*. The converse can be reduced to Fagin's Theorem as follows. We assume that for every structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{N}, W)$ in $\mathcal{K}$, we have that $\max\mathfrak{D} \leq n^k$ where $n = |\mathfrak{D}| = |\mathfrak{A}|$; further we suppose without loss of generality that an ordering $<$ on A is available (otherwise we expand the vocabulary with a binary relation $<$ and add a conjunct $\beta(<)$ asserting that $<$ is a linear order). We can then identify A^k with the initial subset $\{0, \ldots, n^k - 1\}$ of $\mathbb{N}$, viewed as bit positions of the binary representations of the weights of $\mathfrak{D}$. With every $\mathfrak{D} \in \mathcal{K}$ we associate a finite structure $\mathfrak{D}_f$ by expanding the primary part $\mathfrak{A}$ as follows: For every weight function $w \in W$ of arity j we add a new relation P_w of arity $j + k$ with

$$P_w := \{(\bar{a}, \bar{t}) : \text{ the } \bar{t}\text{-th bit of } w(\bar{a}) \text{ is } 1\}.$$

Then $\mathcal{K}$ is in NP if and only if $\mathcal{K}_f = \{\mathfrak{D}_f : \mathfrak{D} \in \mathcal{K}\}$ is an NP-set of finite structures, and in fact, we can choose the encodings in such a way that $\mathfrak{D}$ and $\mathfrak{D}_f$ are represented by the same binary string. Thus, if $\mathcal{K}$ is in NP, then by Fagin's Theorem $\mathcal{K}_f$ is a generalized spectrum, defined by a first-order sentence ψ.

As in Example 3.2, one can construct a first-order sentence α (whose vocabulary consists of the weight functions $w \in \Upsilon_w$ and the corresponding primary relations P_w), which expresses that the P_w encode the weight functions w in the sense defined above. Then $\psi \wedge \alpha$ is a first-order sentence witnessing that $\mathcal{K}$ is a primary metafinite spectrum. □

Remark. The same result holds for simple arithmetical structures.

However, without the restriction that the weights be small, it is no longer true that every NP-set is a primary metafinite spectrum. If we have inputs with huge weights compared to the primary part, then relations over the primary part cannot encode enough information to describe computations that are bounded by a polynomial in the length of the weights.

It is tempting to use unrestricted metafinite spectra instead. However, metafinite spectra in the general sense capture a much larger class than NP.

First, we note that any tuple $\bar{a} \in \mathbb{N}^k$ can be viewed as an arithmetical structure with the empty primary primary vocabulary and k weight functions $a_1, \dots, a_k$ which happen to be nullary. Thus an arithmetical relation $S \subseteq \mathbb{N}^k$ can be viewed as a special class of arithmetical structures.

Theorem 4.7 *Every recursively enumerable set $S \subseteq \mathbb{N}^k$ is a metafinite spectrum. In particular, there exist undecidable metafinite spectra.*

Proof. By Matijasevich's Theorem (see [38]) every recursively enumerable set $S \subseteq \mathbb{N}^k$ is Diophantine, i.e. can be represented as

$$S = \{\bar{a} \in \mathbb{N}^k : \text{ there exists } b_1, \dots, b_m \in \mathbb{N} \text{ such that } Q(\bar{a}, \bar{b}) = 0\}$$

for some polynomial $Q \in \mathbb{Z}[x_1, \dots, x_k, y_1, \dots, y_m]$. Let $P, P' \in \mathbb{N}[\bar{x}, \bar{y}]$ such that $Q(\bar{x}, \bar{y}) = P(\bar{x}, \bar{y}) - P'(\bar{x}, \bar{y})$ Thus S is a metafinite spectrum; the desired first-order sentence uses additional weight functions $b_1, \dots, b_m$ and asserts that $P(\bar{a}, \bar{b}) = P'(\bar{a}, \bar{b})$. □

This can be extended to any r.e. class of arithmetical structures, with arbitrary vocabulary. To prove this, we describe how to encode structures $\mathfrak{D} \subseteq M_\Upsilon[\mathfrak{N}]$ by tuples $c(\mathfrak{D}) \in \mathbb{N}^k$ where k depends only on Υ. (In fact, it is no problem to reduce k to 1.) For future use of such encodings we will be more restrictive than necessary for this result.

Similar to the case of finite structures, an encoding involves the selection of a linear order on the primary part. In fact we find it more convenient to have a *ranking* of the primary part rather than just a linear ordering.

Definition 4.8 Suppose that $\mathfrak{R}$ contains a copy of $(\mathbb{N}, <)$. A *ranking* of a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ is a bijection $r : A \rightarrow \{0, \dots, n-1\} \subseteq R$. A class $\mathcal{K} \subseteq M_\Upsilon[\mathfrak{R}]$ is *ranked* if Υ contains a weight function r whose interpretation on every $\mathfrak{D} \in \mathcal{K}$ is a ranking.

From a ranking one can trivially define a linear order of the primary part. Also a ranking r can be extended to a ranking $r_m : A^m \rightarrow \{0, \dots, n^m - 1\}$ of m-tuples. On the other hand, a ranking need *not* be first-order definable from a linear order; take e.g. $\mathfrak{R} = (\mathbb{N}, <)$. However, if $\sum$ is available then a ranking is definable from a linear order by $r(x) = \sum_y \chi[y < x]$.

We write $\mathcal{R}_\Upsilon$ for the class of *ranked arithmetical structures* of vocabulary Υ.

Lemma 4.9 (Coding Lemma.) *For every vocabulary Υ of ranked arithmetical structures there exists an encoding function*

$$\begin{aligned} c : \mathcal{R}_\Upsilon &\longrightarrow \mathbb{N}^k \\ \mathfrak{D} &\longmapsto c(\mathfrak{D}) = c_1(\mathfrak{D}), \ldots, c_k(\mathfrak{D}) \end{aligned}$$

with the following properties:

(i) c is definable by first-order terms;
(ii) The primary part and the weight functions of $\mathfrak{D}$ can be reconstructured from $c(\mathfrak{D})$ in polynomial time;
(iii) there exists a polynomial $p(n,m)$ such that $c_i(\mathfrak{D}) \leq 2^{p(|\mathfrak{D}|, \max \mathfrak{D})}$ for every $i \leq k$.

Proof. Encode every weight function $w : A^m \to \mathbb{N}$ by a pair (q,s) of natural numbers, where

$$q = \max_{\bar{x}} w(\bar{x}) + 1$$
$$s = \sum_{\bar{x}} w(\bar{x}) q^{r_m(\bar{x})}.$$

This encoding is first-order definable: for q this is obvious, and

$$s = \sum_{\bar{x}} \Big(w(\bar{x}) \prod_{\bar{y}} (q : r_m(\bar{y}) < r_m(\bar{x})) \Big).$$

To encode $\mathfrak{D}$ we pass to the associated algebra $\mathfrak{D}^a$ and represent it by the sequence of pairs (q,s) that encode the weight functions of $\mathfrak{D}^a$. Obviously, properties *(i)*, *(ii)*, *(iii)* are satisfied. □

Theorem 4.10 *Every recursively enumerable class of arithmetical structures is a metafinite spectrum.*

Proof. Let $\mathcal{K} \subseteq M_\Upsilon[\mathfrak{N}]$ be recursively enumerable. Then the set

$$c(\mathcal{K}) := \{c(\mathfrak{D}, r) : \mathfrak{D} \in \mathcal{K},\ r \text{ is a ranking of } \mathfrak{D}\} \subseteq \mathbb{N}^k$$

is also recursively enumerable and therefore Diophantine. The desired first-order sentence ψ uses besides the symbols of Υ a unary weight function r and nullary weight functions $b_1, \ldots, b_m$ and expresses *(i)* that r is a ranking and *(ii)* that $Q(c(\mathfrak{D}, r), \bar{b})) = 0$ for a suitable polynomial $Q \in \mathbb{Z}[x_1, \ldots, x_k, y_1, \ldots, y_m]$ defining $c(\mathcal{K})$. □

Conversely, it is easy to see that every metafinite spectrum of arithmetical structures is recursively enumerable, so we obtain:

Corollary 4.11 *On arithmetical structures, metafinite spectra capture the r.e. sets.*

But there are other contexts where metafinite spectra do indeed capture (a suitable notion of) nondeterministic polynomial-time. An important example are computations over the real numbers with the model of Blum-Shub-Smale.

Theorem 4.12 (Grädel, Meer) $\mathrm{NP}_{\mathbb{R}}$ *coincides with the class of metafinite spectra of* $\mathbb{R}$*-structures.*

The proof is given in [16].

Definition 4.13 Let $\mathcal{K} \subseteq M_{\Upsilon}[\mathfrak{R}]$, and suppose that we have fixed a cost function on R. We say that $\mathcal{K}$ is a *polynomially bounded metafinite spectrum* if there exists a first-order sentence ψ of vocabulary $\Upsilon' \supseteq \Upsilon$ and a polynomial $p(n, m)$ such that $\mathcal{K}$ is the class of all $\mathfrak{D} \in M_{\Upsilon}[\mathfrak{R}]$ for which there exists an expansion $\mathfrak{D}'$ with

- $\mathfrak{D}' \models \psi$
- $\max \mathfrak{D}' \leq p(|\mathfrak{D}|, \max \mathfrak{D})$

Remark. If the cost function is universally bounded by a constant (as in the case of $\mathbb{R}$-structures), then trivially every metafinite spectrum is polynomially bounded.

Conjecture 4.14 *Let* $\mathcal{K} \subseteq M_{\Upsilon}[\mathfrak{N}]$ *be a class of arithmetical structures, which is closed under isomorphism. Then the following are equivalent:*

(i) $\mathcal{K} \in \mathrm{NP}$.
(ii) $\mathcal{K}$ *is a polynomially bounded metafinite spectrum.*

It is not difficult to prove that every polynomially bounded metafinite spectrum is in NP, i.e. that *(ii)* implies *(i)*. The other direction is related to a conjecture of Adleman and Manders concerning the notion of *Diophantine complexity* (see [3, 4, 26, 32, 36, 38]).

Adleman and Manders introduced the class D of all relations $S \subseteq N^k$ that can be represented in the form

$$\bar{a} \in S \Longleftrightarrow \exists y_1 \cdots \exists y_m \Big(\bigwedge_{i=1}^{m} y_i \leq 2^{\max_i \|a_i\|^{\ell}} \wedge Q(\bar{a}, \bar{y}) = 0 \Big)$$

for some $\ell \in \mathbb{N}$ and some polynomial Q with integer coefficients. They conjectured that every arithmetical relation in NP can be given such a Diophantine representation, i.e. that $D = \mathrm{NP}$. A positive solution would imply (and in fact be equivalent to) Conjecture 4.14.

It is obvious that the analogue of Conjecture 4.14 for PTA-structures is true, since there we have all polynomial-time computable functions available. But in fact, much weaker expansions of $\mathfrak{N}_0$ will do as well. Let $\tilde{\mathfrak{N}}$ be obtained from $\mathfrak{N}_0$ by adding at least one of the following functions or relations:

- the so-called *logical and* function, mapping numbers a, b with binary expansions $a = \sum_{i=0}^{m} a_i 2^i$ and $b = \sum_{i=0}^{\ell} b_i 2^i$ to

$$a \& b := \sum_{i=0}^{\min(\ell,m)} \min(a_i, b_i) 2^i.$$

- the partial order $\preceq$ with $a \preceq b$ iff $a\&b = a$, (i.e. every bit of a is less than or equal to the corresponding bit of b);
- the function $(a, b, c) \mapsto \binom{a}{b} \pmod{c}$;
- the modular factorial function $(a, b) \mapsto a! \pmod{b}$.

Then, results of Jones and Matijasevich [32] imply

Theorem 4.15 *Every class in NP of arithmetical structures with secondary part $\tilde{\mathfrak{N}}$ is a polynomially bounded metafinite spectrum.*

The ordering $\preceq$ or the logical and can be directly used to describe computations. Binomial coefficients, and therefore factorials, suffice to define $\preceq$ since $a \preceq b$ if and only if $\binom{a}{b}$ is odd. This follows from Lucas' theorem that, for every prime p, given p-ary representations $a = \sum_i a_i p^i$ and $b = \sum_i b_i p^i$ we have that $\binom{a}{b} = \prod_i \binom{a_i}{b_i} \pmod{p}$.

We can reformulate Theorem 4.15 as follows. If $\mathcal{K}$ is an isomorphism-closed class of arithmetical structures with secondary part $\tilde{\mathfrak{N}}$ (or PTA), then $\mathcal{K}$ is in NP if and only if it can be characterized as the model class of a second-order sentence with bounded quantifiers in the following way:

$$\mathfrak{D} \in \mathcal{K} \ \text{ iff } \ \mathfrak{D} \models (\exists F_1 \leq 2^{p(n,m)}) \cdots (\exists F_k \leq 2^{p(n,m)}) \psi$$

where ψ is first-order and p is a polynomial. Here $(\exists F_i \leq 2^{p(n,m)}) \ldots$ is to be understood as an abbreviation for $\exists F_i [\forall \bar{x} (F_i(\bar{x}) \leq 2^{p(|\mathfrak{D}|, \max \mathfrak{D})}) \wedge \ldots]$.

From results of Hodgson and Kent [26, 36], we obtain a more involved characterization that works also for the secondary part $\mathfrak{N}$, and in fact also for simple arithmetical structures. Here, the second-order prefix has besides the exponentially bounded existential quantifiers $(\exists F_i \leq 2^{p(n,m)})$, also polynomially bounded universal quantifiers of the form $(\forall G_i \leq p(n, m))$. Hodgson and Kent proved that if one generalizes the class D of Adleman and Manders by allowing also polynomially bounded universal quantifiers in the prefix, then one obtains a precise arithmetical characterization of NP. In fact one can even do away with all but one of these universal quantifiers and obtain a normal form which is the analogue to the so-called Davis normal form for r.e sets. The Davis normal form theorem says that every recursively enumerable set $S \subseteq \mathbb{N}^k$ can be represented as

$$S = \{\bar{a} \in \mathbb{N}^k : \exists y_1 (\forall z \leq y_1) \exists y_2 \cdots \exists y_m Q(\bar{a}, \bar{y}, z) = 0\}$$

(where $Q \in \mathbb{Z}[x_1, \ldots, x_k, y_1, \ldots, y_m, z]$); it was an important step towards the eventual solution of Hilbert's 10th problem by Matijasevich. For NP-classes of arithmetical structures this gives the following logical characterization.

Theorem 4.16 *An isomorphism-closed class $\mathcal{K} \subseteq M_\Upsilon[\mathfrak{N}]$ is in NP if and only if there exists a first-order formula ψ and a polynomial $p(n,m)$ such that $\mathcal{K}$ is the class of all $\mathfrak{D} \in M_\Upsilon[\mathfrak{N}]$ with*

$$\mathfrak{D} \models (\exists F_1 \leq 2^{p(n,m)})(\forall G \leq p(n,m))(\exists F_2 \leq 2^{p(n,m)}) \cdots (\exists F_k \leq 2^{p((n,m)})\psi.$$

4.3 Fixed point logics and polynomial-time

Fixed point logics on finite structures. In finite model theory, fixed point logics play a central rôle. They provide a general and flexible method of inductively defining new predicates and thus remedy one of the main deficiencies (with respect to expressiveness) of first order logic: the lack of a mechanism for unbounded recursion or iteration.

We recall the definition of (inflationary) fixed point logic. Let Υ_a be a vocabulary, $R \notin \Upsilon_a$ an r-ary predicate and $\psi(\bar{x})$ a formula of vocabulary $\Upsilon_a \cup \{R\}$ with free variables $\bar{x} = x_1, \ldots, x_r$. Then ψ defines for every finite Υ_a-structure $\mathfrak{A}$ an operator $F_\psi^{\mathfrak{A}} : \mathcal{P}(A^r) \to \mathcal{P}(A^r)$ on the class of r-ary relations over A by

$$F_\psi^{\mathfrak{A}} : R \longmapsto R \cup \{\bar{a} : (\mathfrak{A}, R) \models \psi(\bar{a})\}.$$

By definition, this operator is *inflationary*, i.e $R \subseteq F_\psi^{\mathfrak{A}}(R)$ for all $R \subseteq A^r$. Therefore the inductive sequence $R^0, R^1, \ldots$ defined by $R^0 := \varnothing$ and $R^{j+1} := F_\psi^{\mathfrak{A}}(R^j)$ is increasing, i.e. $R^j \subseteq R^{j+1}$ and therefore reaches a fixed point $R^j = R^{j+1}$ for some $j \leq |A|^r$. It is called the *inflationary fixed point* of ψ on $\mathfrak{A}$, and denoted by R^∞.

Definition 4.17 The (inflationary) fixed point logic FP is defined by adding to the syntax of first order logic the *fixed point formation rule:* if $\psi(\bar{x})$ is a formula of vocabulary $\sigma \cup \{R\}$ as above and $\bar{u}$ is an r-tuple of terms, then

$$[\mathrm{FP}_{R,\bar{x}}\ \psi](\bar{u})$$

is a formula of vocabulary Υ_a, whose semantic is that $\bar{u} \in R^\infty$.

Example 4.18 Here is a fixed point formula that defines the reflexive and transitive closure of the binary predicate E:

$$\mathrm{TC}(u,v) \equiv [\mathrm{FP}_{T,x,y}\ (x = y) \vee (\exists z)(Exz \wedge Tzy)](u,v).$$

Many other variants of fixed point logics have been studied, most notably the *least fixed point logic*, denoted LFP, and the *partial fixed point logic*, denoted PFP. It was proved independently by Immerman [28] and Vardi [48] that, on ordered finite structures, LFP characterizes precisely the queries that are computable in polynomial time. Gurevich and Shelah [23] proved that FP and LFP have the same expressive power on finite structure, so in particular, FP also characterizes PTIME in the presence of a linear ordering. On the class of arbitrary (not necessarily ordered) finite structures, FP and LFP are strictly weaker than PTIME-computability. In fact, on very simple classes of structures, such as structures with the empty vocabulary (i.e. pure sets), FP collapses to first-order logic. Also, the 0-1 law holds for FP, which shows that, on arbitrary finite structures, FP cannot express non-trivial statements about cardinalities.

The fixed point logic FP*. Definition 3.1 gives a general way of extending a logic L for finite structures to a logic L^* of metafinite structures. Applying this definition to FP, we get the logic FP*, the extension of first-order logic FO* by the rule for building fixed point formulae $[\mathrm{FP}_{R,\bar{x}}\ \psi](\bar{u})$ of vocabulary $(\Upsilon_a, \Upsilon_r, \Upsilon_w)$ from already given formulae ψ of vocabulary $(\Upsilon_a \cup \{R\}, \Upsilon_r, \Upsilon_w)$. It is important to emphasize that the inductively defined predicate R is a *predicate over the primary part* and that $\bar{u}$ is a tuple of point terms. We first observe that the fixed point construction preserves PTIME-computability.

Proposition 4.19 *If the basic functions, relations and multiset operations of* $\mathfrak{R}$ *can be evaluated in polynomial time (with respect to the given cost function), then the same is true for all* FP**-definable global function on* $M_\Upsilon[\mathfrak{R}]$.

As in the case of Fagin's Theorem we can also transfer Immerman's and Vardi's logical characterization of PTIME to the case of arithmetical structures with small weights.

Theorem 4.20 *Let* $\mathcal{K} \subseteq \mathcal{R}_\Upsilon$ *be a class of ranked arithmetical structures with small weights. For every global function* G *on* $\mathcal{K}$ *the following are equivalent*

(i) G *is computable in polynomial time.*
(ii) G *is* FP**-definable.*

We omit the proof, which follows by straightforward application of the same arguments as in the proof of Theorem 4.6.

Again, as in the case of metafinite spectra, the restriction to small weights is necessary. For an extreme example, consider polynomial-time predicates $S \subseteq \mathbb{N}$. Each such S gives rise to a decision problem where an instance is an arithmetical structure $\mathfrak{D}$, with a single nullary weight a, and the question is whether $a \in S$. Of course this problem is completely independent of the primary part of the structure, which in particular can be trivial. Fixed point constructions are of absolutely no help here and neither are quantifiers or multiset operations. Thus FP* can decide S if and only if the characteristic function $\chi_S(a)$ is available as a basic term. Obviously there exist polynomial-time predicates S for which this is not the case.

Thus, FP* cannot fully capture PTIME on arithmetical structures, even in the presence of a ranking.

But this is not the only weakness of FP*. Another important limitation is the *absence of any recursion mechanism over numbers and weight functions.* We will exhibit certain interesting consequences of this, by comparing the power of FP* with the fixed point logic with counting (FP + C) on unordered structures. This logic does not include large numbers in the secondary sort, but it has recursion over relations that range over both parts.

Fixed point logic with counting. As we mentioned already in the introduction, among the logics studied in finite model theory, (FP + C) is the closest to our approach. It was first proposed by Immerman, who started from the observation that counting is probably the most basic class of low-complexity queries not expressible in fixed point logic. The original hope was that the addition of counting to FP in a reasonable way should give a logic that could express all of PTIME. It should be pointed out, that there are different ways of adding counting mechanisms to a logic, which are not necessarily equivalent. The most straightforward possibility is the addition of quantifiers of the form $\exists^{\geq 2}$, $\exists^{\geq 3}$, etc., with the obvious meaning. While this is perfectly reasonable for the infinitary logics $L^k_{\infty\omega}$, it is not general enough for fixed point logic, because it does not allow to apply recursion also on the counting parameters i in quantifiers $\exists^{\geq i}x$. In fact if the counting parameters are fixed numbers, then adjoining the quantifiers $\exists^{\geq i}x$ does not give additional power to logics whose formulae may have an arbitrary number of variables (as FO or FP). These counting parameters should therefore be considered as variables that range over the natural numbers. To define in a precise way a logic with counting, and with recursion applicable also to the numbers obtained by counting, it is therefore necessary to extend the original objects of study, namely finite (one-sorted) structures $\mathfrak{A}$ to two-sorted auxiliary structures $\mathfrak{A}^*$ with a second numerical (but also finite) sort.

We are now ready to formally introduce (FP + C). With any one-sorted finite structure $\mathfrak{A}$ one associates the two-sorted structure $\mathfrak{A}^* := (\mathfrak{A}, (\mathbf{n}, <))$ consisting of a copy of $\mathfrak{A}$ for the first sort and the linear order $(\mathbf{n}, <)$ for the second sort, with $n = |A| + 1$ and the standard meaning of $<$ on $\mathbf{n} = \{0, \dots, n-1\}$. We take $n = |A| + 1$ rather than $n = |A|$ to be able to represent the cardinalities of all subsets of $|A|$ within n.

We start with first-order logic over two-sorted vocabularies $(\Upsilon_a, \{<\})$, with the usual semantics over structures $\mathfrak{A}^*$. Latin letters $x, y, z, \dots$ are used for the variables over the first sort, and Greek letters $\lambda, \mu, \nu, \dots$ for variables over the second sort. Note that, contrary to logics of metafinite structures, we have here no restriction on the access of the logic to second sort elements. For instance, we can quantify over number variables to build formulae of the form $\exists\mu\varphi$.

The two sorts are related by *counting terms*, defined by the following rule: Let $\varphi(x)$ be a formula with a free variable x of sort one, then $\#_x[\varphi]$ is a second-sort term, with the set of free variables $\text{free}(\#_x[\varphi]) = \text{free}(\varphi) - \{x\}$. The interpretation of $\#_x[\varphi]$ is the number of first-sort elements a that satisfy $\varphi(a)$. First-order logic with counting, denoted (FO + C), is the closure of two-sorted first-order logic under counting terms.

Example 4.21 To illustrates the use of counting terms we present a formula $\psi(E_1, E_2) \in$ (FO + C) expressing that two equivalence relations E_1 and E_2 over the first sort are isomorphic.

$$\psi(E_1, E_2) := (\forall\mu)(\#_x[\#_y[E_1xy] = \mu] = \#_x[\#_y[E_2xy] = \mu]).$$

The *(inflationary) fixed point logic with counting* (FP + C) is obtained by adding to (FO + C) the mechanism for building fixed point predicates that may range over both sorts.

Definition 4.22 The logic (FP + C), is the closure of two-sorted first-order logic under

(i) the rule for building counting terms;
(ii) the usual rules of first-order logic for building terms and formulae;
(iii) the fixpoint formation rule: Suppose that $\psi(\bar{x}, \bar{\mu})$ is a formula of vocabulary $\Upsilon \cup \{R\}$ where $\bar{x} = x_1, \ldots, x_k$, $\bar{\mu} = \mu_1, \ldots, \mu_\ell$, and R has mixed arity (k, ℓ), and that $(\bar{u}, \bar{\nu})$ is a $k + \ell$-tuple of first- and second-sort terms, respectively. Then

$$[\mathrm{FP}_{R,\bar{x},\bar{\mu}}\ \psi](\bar{u}, \bar{\nu})$$

is a formula of vocabulary Υ.

The semantics of $[\mathrm{FP}_{R,\bar{x},\bar{\mu}}\ \psi]$ on $\mathfrak{A}^*$ is defined in the same way as for the logic FP, namely as the inflationary fixed point R^∞ of the operator

$$F_\psi^{\mathfrak{A}^*} : R \longmapsto R \cup \{(\bar{u}, \bar{\nu}) \mid (\mathfrak{A}^*, R) \models \psi(\bar{u}, \bar{\nu})\}.$$

(FP + C) was first introduced by Immerman, in a different but equivalent form, with counting quantifiers rather than counting terms. The present version appeared first in [17].

Example 4.23 An interesting example for an (FP + C)-computable global function is the *stable colouring* of a graph. Given a graph G with a colouring $f : V \to 0, \ldots, r$ of its vertices, we define a refinement f' of f, where vertex x has the new colour $f'x = (fx, n_1, \ldots, n_r)$ where $n_i = \#y[Exy \wedge (fy = i)]$. The new colours can be sorted lexicographically so that they form again an initial subset of $\mathbb{N}$. Then the process can be iterated until a fixed point, the stable colouring of G is reached. It is known that almost all graphs have the property that no two vertices have the same stable colour. Thus stable colourings provide a polynomial-time graph-canonization algorithm for a dense class of graphs. It should be clear that the stable colouring of a graph is definable in (FP + C) (see [31] for more details).

Over arithmetical structures, we can define counting in FO* and hence FP*, as shown in Example 3.3. One might therefore feel that FP*, having both a fixed point constructor and the ability to count, is at least as powerful as (FP + C).

To make this a precise question, we have to consider a setting where the two logics can be compared. We compare their expressive powers on classes $\mathcal{K} \subseteq \mathrm{Fin}(\Upsilon_a)$ of finite, one-sorted structures.

Definition 4.24 With every finite structure $\mathfrak{A}$ and every secondary part $\mathfrak{R}$ we associate the metafinite structure $\mathfrak{A}_\mathfrak{R} := (\mathfrak{A}, \mathfrak{R}, \varnothing)$, with primary part $\mathfrak{A}$, secondary part $\mathfrak{R}$ and the empty set of weight functions. We say, that a model class $\mathcal{K} \subseteq \mathrm{Fin}(\Upsilon_a)$ of finite structures is FP*-definable over $\mathfrak{R}$, if there exists a sentence $\psi \in \mathrm{FP}^*$ such that

$\mathcal{K} = \{\mathfrak{A} \in \mathrm{Fin}(\Upsilon_a) : \mathfrak{A}_{\mathfrak{N}} \models \psi\}.$

As usual we say that $\mathcal{K}$ is (FP + C)-definable if there exists a sentence $\theta \in$ (FP + C) such that

$\mathcal{K} = \{\mathfrak{A} \in \mathrm{Fin}(\Upsilon_a) : \mathfrak{A}^* \models \theta\}.$

Proposition 4.25 *Let $\mathfrak{N}$ be any reduct of* PTA. *Then every model class $\mathcal{K} \subseteq \mathrm{Fin}(\Upsilon_a)$ which is* FP**-definable over $\mathfrak{N}$, is also* (FP + C)*-definable.*

Proof. This follows by straightforward induction over terms and formulae of FP*, using the facts that *(i)* every FP*-definable global function can be evaluated in polynomial time and that *(ii)* every polynomial-time computable function or relation appearing in the secondary part can be expressed by an (FP + C)-definition over the numerical sort (since the numerical sort is ordered). □

The converse is not always true. Indeed, let $\mathfrak{N} = \mathfrak{N}_0$. If we consider the case that $\Upsilon_a = \varnothing$, then, by taking cardinalities, a class $\mathcal{K} \subseteq \mathrm{Fin}(\varnothing)$ can be viewed as a set of natural numbers. On $\mathrm{Fin}(\varnothing)$, (FP + C) captures polynomial-time with respect to the cardinality of the structures, i.e. $\mathcal{K} \subseteq \mathrm{Fin}(\varnothing)$ is (FP + C)-definable if and only if $\{1^n : n \in \mathcal{K}\}$ is decidable in polynomial time. On the other hand, FP* on structures $(A, \mathfrak{N}_0, \varnothing)$ is equivalent to FO* whose power can be precisely described as follows: Every sentence φ can be written as a Boolean combination of inequalities $f(n) \leq g(n)$ where $f, g \in T$ are terms in one variable n that represents the cardinality of A. Since all elements of A are indistinguishable, the terms $\sum_x F$ or $\prod_x F$ produced by means of the multiset operations can simple be rewritten as $n \cdot F$ and F^n respectively. (Applications of max and min have no effect at all.) Thus the set T of terms can be defined by closing the constants and n under addition, multiplication and under raising to nth power (i.e. given $t(n)$, one can form $t(n)^n$). A simple diagonalization arguments proves that there exist predicates $S \subseteq \mathbb{N}$ which cannot be defined in this way, but nevertheless $\{1^n : n \in S\}$ is decidable in polynomial time.

Indeed, let φ be a Boolean combination of inequalities $f \leq g$ with $f, g \in T$. Syntactically, φ is a string in a finite alphabet whose symbols are $0, 1, n, +, \cdot$, etc. We can order this alphabet and assign numbers to strings in the usual way. Let $n(\varphi)$ be the number associated with φ and S be the set of those numbers $n(\varphi)$ such that φ is false at $n(\varphi)$. Clearly, S is not defined by any φ. Moreover, since φ is equivalent to $(\varphi \wedge 0 < 1)$, $(\varphi \wedge 0 < 1 \wedge 0 < 1)$ etc., φ differs from S on infinitely many numbers.

It thus suffices to prove that there exists a polynomial-time algorithm that, given $1^{n(\varphi)}$, computes the truth value of φ at $n(\varphi)$ and inverts the result. This is obvious, once we have checked, by an easy induction on the formation rules of T, that for every term $f \in T$, the logarithm of the value $f(n)$ is bounded by a polynomial in n.

We thus have proved the following result.

Proposition 4.26 *There exist model classes $\mathcal{K}$ of finite structures which are* (FP + C)*-definable, but not* FP**-definable over $\mathfrak{N}_0$.*

The fact that $\mathfrak{N}_0$ forms a counterexample to the converse of Proposition 4.25 survives various enrichments of $\mathfrak{N}_0$. In fact, the same proof works if $\mathfrak{N}_0$ is extended by any finite collection of polynomial-time computable functions and any finite collection of multiset operations Γ such that the value of Γ at multisets $\{\!\{t, t, \dots, t\}\!\}$, consisting of n occurrences of t, can be computed in polynomial-time with respect to n and $\log t$. However, there is a limit to such generalizations. We will prove in Sect. 5 that the converse of Proposition 4.25 does hold in the case that $\mathfrak{N} = \text{PTA}$.

Remark. Note that the problem of capturing polynomial-time on *ranked* PTA-structures is trivial and does not require a fixed-point construction. As pointed out above, if a ranking is available, then the primary part can be encoded by a tuple of natural numbers and this encoding is definable by first-order terms. Any polynomial-time property is thus reducible to a PTIME property of numbers which is a basic relation of PTA. Thus a global function on ranked PTA-structures is PTIME-computable if and only if it is first-order definable. Furthermore FO* and FP* coincide on ranked PTA-structures.

4.4 A functional fixed point logic

One possibility to overcome the limitations of languages of type L^* is to apply recursion in one way or another to weight functions.

We discuss here, as one particular example, a *fixed-point calculus* for *partially defined weight functions.* It is convenient to deal with partial functions by extending the secondary part $\mathfrak{R}$ by a new element to a structure $\mathfrak{R}^*$ with universe $R \cup \{\text{undef}\}$ in the following way:

The relations of $\mathfrak{R}^*$ coincide with their restrictions to $\mathfrak{R}$ and the functions and multiset operations of $\mathfrak{R}$ are extended to $\mathfrak{R}^*$ in some arbitrary way. For many functions, the natural choice will be to set $f^{\mathfrak{R}^*}(\bar{a}) = \text{undef}$ whenever the argument $\bar{a}$ contains undef. However, for some functions there are other reasonable possibilities: For multiplication, it actually makes more sense to set

$$a \cdot \text{undef} = \text{undef} \cdot a = \begin{cases} 0 & \text{if } a = 0 \\ \text{undef} & \text{if } a \neq 0. \end{cases}$$

Fix a signature Υ and a function symbol Z not contained in Υ. Let $G(Z, \bar{x})$ be a weight term of signature $(\Upsilon_a, \Upsilon_r, \Upsilon_w \cup \{Z\})$ and free variables $\bar{x} = x_1, \dots, x_r$ where r is the arity of Z. We write $G^{\mathfrak{D},Z}(\bar{x})$ for the value of $G(Z, \bar{x})$ for a given interpretation $(\mathfrak{D}, Z)$.

For every structure $\mathfrak{D} \in M_\Upsilon[\mathfrak{R}^*]$, the term $G(Z, \bar{x})$ gives rise to an operator $F_G^{\mathfrak{D}}$ which updates *partially defined* functions Z as follows:

$$F_G^{\mathfrak{D}}(Z)(\bar{a}) := \begin{cases} G^{\mathfrak{D},Z}(\bar{a}) & \text{if } Z(\bar{a}) = \text{undef} \\ Z(\bar{a}) & \text{otherwise.} \end{cases}$$

This gives an inductive definition of a sequence of partial weight functions $Z^j : A^r \to \mathbb{R}$.

Z^0 is undefined everywhere (i.e. $Z^0(\bar{a}) = \text{undef}$ for all $\bar{a}$)
$Z^{j+1} = F_G^{\mathfrak{D}}(Z^j)$.

The operator $F_G^{\mathfrak{D}}$ updates Z only at points where Z is undefined, so this process reaches a fixed point after a polynomial number of iterations: $Z^j = Z^{j+1}$ for some $j \leq |A|^r$. We denote this fixed point by Z^∞ and call it the *fixed point of* $G(Z, \bar{x})$ *on* $\mathfrak{D}$.

Definition 4.27 *Functional fixed point logic*, denoted FFP, is obtained by augmenting the first-order term calculus FOT (see Definition 3.8) with the following rule for building terms:

If $G(Z, \bar{x})$ is a weight term of signature $(\Upsilon_a, \Upsilon_r, \Upsilon_w \cup \{Z\})$, if $\bar{x} = x_1, \ldots, x_r$ is a tuple of variables (where r is the arity of Z), and if $\bar{u} = u_1, \ldots, u_r$ is a tuple of point terms, then

$$\mathbf{fp}[Z(\bar{x}) \leftarrow G(Z, \bar{x})](\bar{u})$$

is a weight term of signature $(\Upsilon_a, \Upsilon_r, \Upsilon_w)$. Its semantic, on a given structure $\mathfrak{D}$, is $Z^\infty(\bar{u})$.

Note, that on arithmetical structures, FFP can define weights of double exponential size. Indeed suppose we have arithmetical structures with a ranking r and let us adopt the conventions that max and $+$ produce undef whenever any of the arguments is undefined, and that $0 \cdot \text{undef} = 0$. Set

$$G(Z, x) := 2\chi[r(x) = 0] + \max_y \Big(\chi[r(x) = r(y) + 1] \prod_z Z(y)\Big).$$

Then, for every structure $\mathfrak{D}$ with $|\mathfrak{D}| = n$ we have that

$$\mathbf{fp}[Z(x) \leftarrow G(Z, x)](y) = 2^{n^{r(y)}}.$$

This even works for simple arithmetical structures, because the term $\prod_z Z(y)$ — which evaluates to $Z(y)^n$ — can be simulated by a fixed point construction.

However, in the context of computations over $\mathbb{R}$ with the Blum-Shub-Smale model, the magnitude of the numbers is no serious problem, since one assumes unit cost for each $r \in \mathbb{R}$. In fact it has been shown in [16] that functional fixed-point logic is the right logic for describing polynomial-time computability in that model, in the sense that it gives rise to the following analogue of the Immerman-Vardi Theorem.

Theorem 4.28 (Grädel, Meer) *On ranked* $\mathbb{R}$*-structures,* FFP *captures* $\mathrm{P}_{\mathbb{R}}$.

Remark. For some applications the update operator $F_G^{\mathfrak{D}}$ used for FFP may not seem adequate, since the values different from undef are never updated. Instead we may consider a different update operator $\tilde{F}_G^{\mathfrak{D}}$ with

$$\tilde{F}_G^{\mathfrak{D}}(Z)(\bar{a}) := G^{\mathfrak{D}, Z}(\bar{a}).$$

Of course, the inductive process defined by such an operator need not reach a fixed point. But — as in the case of the partial fixed point logic PFP considered in finite model theory — we can define Z^∞ to be the fixed point of the sequence $Z^0, Z^1, \ldots$, defined by $\tilde{F}_G^{\mathfrak{D}}$, if the fixed point exists, and some default value, e.g. the everywhere undefined function, otherwise.

We don't further investigate this approach here. The study of this, and related variants of functional fixed point logics, as well as other means of inductive definability of weight functions, is one of the promising directions for future research.

5 Back and forth from finite to metafinite structures

As explained in the introduction, our goal is to extend the approach and methods of finite model theory to the more general class of metafinite structures. We show in this section that an important methodology of finite model theory, namely the various variants of Ehrenfeucht-Fraïssé games, is indeed applicable in our more general context.

The aspect that we consider here is the indistinguishability of two metafinite structures by (infinitary) logics with bounded variables, but with arbitrary multiset operations. We show that this reduces to the indistinguishability of two associated *finite* structures by *first-order* formulae *with counting.*

Throughout this section, we consider structures with a fixed secondary part $\mathfrak{R}$ and assume that the primary part is always relational.

5.1 Indistinguishability by logics with k variables

Definition 5.1 Let $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ and $\mathfrak{D}' = (\mathfrak{B}, \mathfrak{R}, W')$ be structures in $M_\Upsilon(\mathfrak{R})$, let $\bar{a}$ and $\bar{b}$ be ℓ-tuples of elements of $\mathfrak{A}$ and $\mathfrak{B}$ respectively, and let L be a logic of metafinite structures. We say that $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$ are L-equivalent — in symbols: $(\mathfrak{D}, \bar{a}) \equiv_L (\mathfrak{D}', \bar{b})$ — if for every weight term $F(x_1, \ldots, x_\ell)$ of L,

$$F^{\mathfrak{D}}(\bar{a}) = F^{\mathfrak{D}'}(\bar{b}).$$

Since in our logics we have for every formula its characteristic function available as a weight term, the L-equivalence of $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$ implies in particular that for every formula $\varphi(\bar{x})$ of L

$$\mathfrak{D} \models \varphi(\bar{a}) \text{ if and only if } \mathfrak{D}' \models \varphi(\bar{b}).$$

The converse does not necessarily hold, i.e., two structures may be indistinguishable by formulae of L but there nevertheless may exist a weight term that separates them. This may be the case when $\mathfrak{R}$ contains unreachable elements which do not appear as values of any closed Υ_r-term.

Logics with k variables. We first recall the definitions of some logics with bounded number of variables that are of great importance in finite model theory.

L^k is the fragment of first-order logic consisting of the formulae whose variables, both free and bound, are among $x_1, \ldots, x_k$. The infinitary logic $L^k_{\infty\omega}$ is the closure of L^k under conjunctions and disjunctions applied to arbitrary sets of formulae. Further, $L^\omega_{\infty\omega} = \bigcup_{k\in\omega} L^k_{\infty\omega}$. It is well-known that the familiar fixed point logics LFP, IFP and PFP are sublogics of $L^\omega_{\infty\omega}$.

The logics C^k, $C^k_{\infty\omega}$ and $C^\omega_{\infty\omega}$ are the extension of L^k, $L^k_{\infty\omega}$ and $L^\omega_{\infty\omega}$ by means of counting quantifiers $\exists^{\geq 2}$, $\exists^{\geq 3}$, etc., with the obvious semantic. One of the reasons why these logics are important is that $C^\omega_{\infty\omega}$ is an extension of fixed point logic with counting (FP + C).

Equivalence with respect to $L^k_{\infty\omega}$ has an elegant characterization in terms of the k-pebble game [6, 27, 43], an infinitary variant of Ehrenfeucht-Fraïssé games. There is a similar pebble game appropriate to $C^k_{\infty\omega}$ [31]. It is played by two players, I and II, on two structures $\mathfrak{A}$ and $\mathfrak{B}$ of the same relational signature. They have k pairs of pebbles.

A move of the game is played as follows.

1. Player I chooses $i \leq k$ and picks up the i-th pair of pebbles. He selects a nonempty subset X of either A or B. Player II chooses a subset Y in the other structure with $|Y| = |X|$. If no such set exists, the game is over and Player I has won.
2. Player I places an i-pebble on an element $y \in Y$. Player II puts the other i-pebble on an element $x \in X$.

After any move, the pebbles on the 'board' define a partial map from A to B, taking every pebbled element of A to the element of B carrying the corresponding pebble. Player II has to maintain the condition that the pebble map is a partial isomorphism. We say that Player II wins the C^k-game on $(\mathfrak{A}, a_1, \ldots, a_\ell)$ and $(\mathfrak{B}, b_1, \ldots, b_\ell)$ if she has a strategy to maintain this condition forever, when initially the first ℓ pairs of pebbles are placed on $(a_1, b_1), \ldots, (a_\ell, b_\ell)$.

Theorem 5.2 (Immerman, Lander) *The following are equivalent*

(i) Player II wins the C^k-game on $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$.
(ii) $\mathfrak{A} \models \varphi(\bar{a})$ iff $\mathfrak{B} \models \varphi(\bar{b})$ for every formula $\varphi(\bar{x}) \in C^k_{\infty\omega}$.

Here is another way to put and to refine this (see [17, 41]). For a tuple $\bar{a} \in (A \cup \{*\})^k$ (where $*$ serves as a dummy value in the case that not all k variables are actually used) we write $\bar{a}\frac{c}{j}$ for the tuple obtained by substituting (or adding) c at position j to $\bar{a}$.

We write $(\mathfrak{A}, \bar{a}) \sim_i (\mathfrak{B}, \bar{b})$ if Player II has a strategy to maintain the winning condition for at least i moves of the C^k-game, starting at position $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{B}, \bar{b})$. Note that $(\mathfrak{A}, \bar{a}) \sim_0 (\mathfrak{B}, \bar{b})$ if and only if $p : \bar{a} \longmapsto \bar{b}$ is a partial isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$.

Theorem 5.3 $(\mathfrak{A}, \bar{a}) \sim_{i+1} (\mathfrak{B}, \bar{b})$ *if and only if* $(\mathfrak{A}, \bar{a}) \sim_i (\mathfrak{B}, \bar{b})$, *and for every $\sim_i$-equivalence class C and every $j \leq k$ we have that*

$$\#\{c \in A : (\mathfrak{A}, \bar{a}\tfrac{c}{j}) \in C\} = \#\{d \in B : (\mathfrak{B}, \bar{b}\tfrac{d}{j}) \in C\}.$$

Since $C^k_{\infty\omega}$-equivalence is the intersection of all equivalence relations $\sim_i$ one obtains the following characterization.

Theorem 5.4 *$C^k_{\infty\omega}$-equivalence is the coarsest equivalence relation $\sim$ with the following properties If $(\mathfrak{A}, \bar{a}) \sim (\mathfrak{B}, \bar{b})$ then*

(i) The function $p : \bar{a} \longmapsto \bar{b}$ is a partial isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$;
(ii) for every $\sim$-equivalence class C and every $j \leq k$ we have that

$$\#\{c \in A : (\mathfrak{A}, \bar{a}\tfrac{c}{j}) \in C\} = \#\{d \in B : (\mathfrak{B}, \bar{b}\tfrac{d}{j}) \in C\}.$$

An infinitary k-variable term calculus for metafinite structures with multiset operations. $L_{\infty\omega}$ generalizes first-order logic. In a similar vein, we generalize the first-order term calculus FOT(Υ) given by Definition 3.8 and the subsequent remark (because Υ_a is not necessary empty). Let FOT$^k(\Upsilon)$ be the restriction of FOT(Υ) to terms using only the variables $x_1, \ldots, x_k$.

Define a set operation on a set R to be a unary operation from subsets of R to R. Let Υ^* be the extension of Υ with names for all multiset operations over R, and let $\mathfrak{R}^*$ be the corresponding expansion of $\mathfrak{R}$.

Definition 5.5 The term calculus $T^k_{\infty\omega}(\Upsilon, R)$ is the extension of the term calculus FOT$^k(\Upsilon^*)$ with the following rule: If S is a set operation on R and Φ is a set (any set) of terms, then $S(\Phi)$ is a term. The rank of $S(\Phi)$ is the supremum of the ranks of terms in Φ (which may be an infinite ordinal). The semantics is as follows: Given an evaluation of the variables, compute the set $X \subseteq R$ of the values of terms in Φ under that evaluation, and then apply S to X.

Remark. The relation of F being a proper subterm of a term G is well founded.

Remark. Let us see that the characteristic function of every $C^k_{\infty\omega}$ formula φ about the primary part is given by some term t_φ in $T^k_{\infty\omega}$. The characteristic functions of the primary relations are always available. If $\varphi = \neg\psi$ then the desired $t_\varphi = S(\{t_\psi\})$ where S is any set operation such that $S(\{0\}) = 1$ and $S(\{1\}) = 0$. If φ is a disjunction of formulas φ_i where $i \in I$ then $t_\varphi = S(\{t_{\varphi_i} : i \in I\})$ where S is any operation that coincides with max on nonempty subsets of $\{0, 1\}$. To handle counting quantifiers, let Γ^i be a multiset operation such that $\Gamma^i(m) = 1$ if m contains at least i occurrences of 1 and $\Gamma^i(m) = 0$ otherwise. If $\psi = \exists^{\geq i} x\varphi$ then $t_\psi = (\Gamma^i)_x t_\varphi$.

Example 5.6 Suppose that a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{N}, W)$ is such that every element $a \in A$ is definable in $\mathfrak{A}$ by some formula $\varphi_a(x)$, and W contains a unary weight function w. Let S be a set function such that $S(X) = 1$ if and only if every number in X is prime. The the term

$$S(\{\chi[\varphi_a](x) \cdot w(x) : a \in A\})$$

evaluates to 1 in $\mathfrak{D}$ if and only if the range of w consists of primes.

Remark. In the remainder of this section, we prove various theorems about the term calculus $T^k_{\infty\omega}$. The developed theory is quite robust with respect to the definition of $T^k_{\infty\omega}$. It does not change if the $T^k_{\infty\omega}$ is further enriched by means of even fancier super-operations over R; for example we may require that, for every finitary or infinitary operation $f(r_1, r_2, \ldots)$ over R and terms $t_i \in T^k_{\infty\omega}$, the possibly infinitary expression $f(t_1, t_2, \ldots)$ is a term in $T^k_{\infty\omega}$. On the other hand, as the remark above shows, we actually use only very simple set operations.

5.2 Partial isomorphisms and the multiset pebble game.

Consider a metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W) \in M_\Upsilon(\mathfrak{R})$. We associate with $\mathfrak{D}$ a finite structure $\text{fin}(\mathfrak{D})$ with universe A, by expanding $\mathfrak{A}$ with relations

$$P_{w,r} := \{\bar{a} : w^{\mathfrak{D}}(\bar{a}) = r\}$$

for every function $w \in W$ and every element $r \in R$. Although the set of these new predicates is infinite, only finitely many relations are nonempty.

Definition 5.7 Let $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ and $\mathfrak{D}' = (\mathfrak{B}, \mathfrak{R}, W')$ belong to $M_\Upsilon(\mathfrak{R})$. A *partial isomorphism* from $\mathfrak{D}$ to $\mathfrak{D}'$ is a function $p : A_0 \to B$ whose domain is $A_0 \subseteq A$ such that

- for every relation symbol $R \in \Upsilon_a$ and all elements $a_1, \ldots, a_m \in A_0$

$$\mathfrak{D} \models R(a_1, \ldots, a_m) \text{ if and only if } \mathfrak{D}' \models R(pa_1, \ldots, pa_m).$$

- for every function symbol $w \in \Upsilon_w$ and all elements $a_1, \ldots, a_m \in A_0$ we have that

$$w^{\mathfrak{D}}(a_1, \ldots, a_m) = w^{\mathfrak{D}'}(pa_1, \ldots, pa_m).$$

Thus, the partial isomorphisms from $\mathfrak{D}$ to $\mathfrak{D}'$ are precisely the partial isomorphisms from $\text{fin}(\mathfrak{D})$ to $\text{fin}(\mathfrak{D}')$.

We now describe the 'obvious' pebble game appropriate to the logic $T^k_{\infty\omega}$. Given two metafinite structures $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ and $\mathfrak{D}' = (\mathfrak{B}, \mathfrak{R}, W')$ in $M_\Upsilon(\mathfrak{R})$, the T^k-game on $(\mathfrak{D}, \mathfrak{D}')$ is played with k pairs of pebbles on the 'board' (A, B). A move of the T^k-game is played as follows:

1. Player I selects $\ell \leq k$ pairs of pebbles and selects a function $f : A^\ell \to R$. Player II chooses a function $g : B^\ell \to R$ such that $\text{mult}(f) = \text{mult}(g)$. (Recall that $\text{mult}(f) = \{\!\{f(\bar{a}) : \bar{a} \in A^\ell\}\!\}$.) If no such function exists, the game is over and Player I has won.
2. Player I puts the selected pebbles on elements $b_1, \ldots, b_\ell \in B$. Player II puts the corresponding pebbles on $a_1, \ldots, a_\ell$ such that $f(a_1, \ldots, a_\ell) = g(b_1, \ldots, b_\ell)$.

Remark. It might seem that there is an asymmetry here, since Player I always selects a function on the first structure and always pebbles elements on the second one, and that instead, he should be allowed to choose on which structure he defines a function. However, this would not change the game in an essential way. The condition that Player II answers with a function defining the same multiset is very restrictive and makes it unnecessary to let Player I choose the structure first. In particular, I wins immediately if the primary parts of the two structures do not have the same cardinality. It should be noted that if two structures $\mathfrak{A}$ and $\mathfrak{B}$ are known to have the same number of elements, then also the C^k-game on $\mathfrak{A}$ and $\mathfrak{B}$ can be restricted such that Player I always chooses his sets in $\mathfrak{A}$ and pebbles elements of $\mathfrak{B}$, but never vice versa.

The moves in the T^k-game simulate the use of the multiset operations. However, it turns out that the T^k-game is equivalent to the C^k-game of Immerman and Lander. We prove this by way of two Lemmata.

Lemma 5.8 *Let $F(\bar{x})$ be a weight term in $T^k_{\infty\omega}$ of rank α such that $F^{\mathfrak{D}}(\bar{a}) \neq F^{\mathfrak{D}'}(\bar{b})$. Then*

(i) Player I wins the C^k-game on $(\text{fin}(\mathfrak{D}), \bar{a})$ and $(\text{fin}(\mathfrak{D}'), \bar{b})$. Furthermore if α is finite then he wins the game in α moves.

(ii) Player I wins the T^k-game on $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$. Furthermore if α is finite then he wins the game in α moves.

Proof. Obviously, *(i)* implies *(ii)*. We prove *(i)* by induction on α, the case $\alpha = 0$ being trivial. Let $F^{\mathfrak{D}}(\bar{a}) \neq F^{\mathfrak{D}'}(\bar{b})$ for some term F of rank $\alpha > 0$. If $F = S(\Phi)$ for some operation S and set of terms Φ, then $G^{\mathfrak{D}}(\bar{a}) \neq G^{\mathfrak{D}'}(\bar{b})$ for at least one $G \in \Phi$; similarly, if $F = g(F_1, \ldots, F_m)$ then at least one subterm F_i separates $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$.

Since the process of descending to proper subterms is well-founded, F contains at least one subterm separating $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$ which either is of rank zero, in which case we are done, or of the form

$$\Gamma_{\bar{y}}(G(\bar{x}, \bar{y}) : H(\bar{x}, \bar{y}) = 1)$$

where G and H have ranks $< \alpha$. For ease of notation, we assume that $\bar{x}$ and $\bar{y}$ are disjoint tuples of variables among $x_1, \ldots, x_k$. In the case of finite α, the ranks of G and H are bounded by $\alpha - \ell$ where ℓ is the length of $\bar{y}$.

Thus, G and H define distinct multisets on the two structures:

$$\{\!\{G^{\mathfrak{D}}(\bar{a}, \bar{c}) : \bar{c} \in A^\ell, H^{\mathfrak{D}}(\bar{a}, \bar{c}) = 1\}\!\} \neq \{\!\{G^{\mathfrak{D}'}(\bar{b}, \bar{d}) : \bar{d} \in B^\ell, H^{\mathfrak{D}'}(\bar{b}, \bar{d}) = 1\}\!\}.$$

As a consequence there exists $r \in R$ such that

$$\#\{\bar{c} \in A^\ell : G^{\mathfrak{D}}(\bar{a}, \bar{c}) = r \wedge H^{\mathfrak{D}}(\bar{a}, \bar{c}) = 1\}$$

$$\neq \#\{\bar{d} \in B^\ell : G^{\mathfrak{D}'}(\bar{b}, \bar{d}) = r \wedge H^{\mathfrak{D}'}(\bar{b}, \bar{d}) = 1\}.$$

This implies that there exist natural numbers $m_1, \ldots, m_\ell$ such that

$$\exists^{\geq m_1} y_1 \cdots \exists^{\geq m_\ell} y_\ell [G^{\mathfrak{D}}(\bar a, \bar y) = r \wedge H^{\mathfrak{D}}(\bar a, \bar y) = 1] \text{ but}$$

$$\textbf{not } \exists^{\geq m_1} y_1 \cdots \exists^{\geq m_\ell} y_\ell [G^{\mathfrak{D}'}(\bar b, \bar y) = r \wedge H^{\mathfrak{D}'}(\bar b, \bar y) = 1]$$

(or vice versa). Player I wins by the following strategy: in his first ℓ moves he selects appropriate sets $A_1, \ldots, A_\ell \subseteq A$ of cardinalities $m_1, \ldots, m_\ell$ so that $G^{\mathfrak{D}}(\bar a, \bar c) = r$ and $H^{\mathfrak{D}}(\bar a, \bar c) = 1$ for the tuples $\bar c = c_1, \ldots, c_\ell$ with $c_i \in A_i$. By induction on ℓ it follows easily that whatever sets $B_1, \ldots B_\ell \subseteq B$ are chosen by Player II in these first ℓ moves, Player I can pebble elements $d_1, \ldots, d_\ell$ such that $G^{\mathfrak{D}'}(\bar b, \bar d) \neq r$ or $H^{\mathfrak{D}'}(\bar b, \bar d) \neq 1$. Since both G and H have ranks $< \alpha$, the induction hypothesis implies that Player I wins the remaining game, and, in the case of finite α, that he wins the remaining game in $\alpha - \ell$ moves. □

Lemma 5.9 *If Player II wins the C^k-game on* $(\text{fin}(\mathfrak{D}), \bar a)$ *and* $(\text{fin}(\mathfrak{D}'), \bar b)$, *then she also wins the T^k-game on* $(\mathfrak{D}, \bar a)$ *and* $(\mathfrak{D}', \bar b)$.

Proof. For fixed structures $\mathfrak{D}$, $\mathfrak{D}'$, the positions in both games are given by the tuples $\bar a, \bar b$ of pebbled elements. Since the winning conditions of the two games are identical it suffices to show the following: Suppose that Player II has a winning strategy for the C^k-game from position $(\bar a, \bar b)$. Then Player II has a strategy for *one move of the T^k-game* from position $(\bar a, \bar b)$ to reach a position from which she again has a *winning strategy for the C^k-game*. It then follows that also in the T^k-game, Player II can forever maintain the condition that the pebbled elements define a partial isomorphism between the primary parts.

Suppose that Player I, in the T^k-game from position $(\bar a, \bar b)$, starts by selecting pebbles $j_1, \ldots, j_\ell$ and defining a function $f : A^\ell \longrightarrow R$. By the assumption, Player II wins the C^k-game from $(\bar a, \bar b)$. Thus $(\text{fin}(\mathfrak{D}), \bar a) \equiv_{C^k_{\infty\omega}} (\text{fin}(\mathfrak{D}'), \bar b)$. By Theorem 5.4, this implies that, for for every $C^k_{\infty\omega}$-equivalence class C and every $j \leq k$, we have that

$$\#\{c \in A : (\text{fin}(\mathfrak{D}), \bar a\tfrac{c}{j}) \in C\} = \#\{d \in B : (\text{fin}(\mathfrak{D}'), \bar b\tfrac{d}{j}) \in C\}.$$

Repeating the argument, we get that for every equivalence class C and every $\bar{\jmath} = j_1, \ldots, j_\ell$

$$\#\{\bar c \in A^\ell : (\text{fin}(\mathfrak{D}), \bar a\tfrac{\bar c}{\bar{\jmath}}) \in C\} = \#\{d \in B^\ell : (\text{fin}(\mathfrak{D}'), \bar b\tfrac{\bar d}{\bar{\jmath}}) \in C\}.$$

Thus, there exists a bijection $\pi : A^\ell \rightarrow B^\ell$ such that for all $\bar c \in A^\ell$

$$(\text{fin}(\mathfrak{D}), \bar a\tfrac{\bar c}{\bar{\jmath}}) \equiv_{C^k_{\infty\omega}} (\text{fin}(\mathfrak{D}'), \bar b\tfrac{\pi\bar c}{\bar{\jmath}}).$$

Now, Player II defines $g : B^\ell \rightarrow R$ as $g := f \circ \pi$, and, if Player I pebbles $\bar d \in B^\ell$, she answers with the unique tuple $\bar c \in A^\ell$ such that $\pi\bar c = \bar d$. The resulting positions are in the same $C^k_{\infty\omega}$-equivalence class, so Player II has a again reached a winning position. □

Thus, we have established the following result.

Theorem 5.10 *Let $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, W)$ and $\mathfrak{D}' = (\mathfrak{B}, \mathfrak{R}, W')$ be structures in $M_\Upsilon(\mathfrak{R})$ and $\bar{a}$ and $\bar{b}$ be ℓ-tuples of elements of $\mathfrak{A}$ and $\mathfrak{B}$, respectively. The following are equivalent*

(i) Player II wins the T^k-game on $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$.
(ii) $(\mathfrak{D}, \bar{a})$ and $(\mathfrak{D}', \bar{b})$ are $T^k_{\infty\omega}$-equivalent.
(iii) Player II wins the C^k-game on $(\mathrm{fin}(\mathfrak{D}), \bar{a})$ and $(\mathrm{fin}(\mathfrak{D}'), \bar{b})$.
(iv) $(\mathrm{fin}(\mathfrak{D}), \bar{a})$ and $(\mathrm{fin}(\mathfrak{D}'), \bar{b})$ are $C^k_{\infty\omega}$-equivalent.

5.3 Invariants

The descriptions of $L^k_{\infty\omega}$- or $C^k_{\infty\omega}$-equivalence in terms of the k-pebble games give rise to invariants that represent in a compact way, by means of an *ordered finite structure*, the complete $L^k_{\infty\omega}$- or $C^k_{\infty\omega}$-theory of a given finite structure.

The first such invariants were found by Abiteboul and Vianu [2]. They were formulated in terms of computability by relational machines rather than $L^k_{\infty\omega}$-definability, but the notions are very closely related. With these invariants, Abiteboul and Vianu could prove that the logics FP and PFP coincide (with respect to expressive power) if and only if PTIME = PSPACE. We refer to [14] for a very nice exposition in terms of $L^k_{\infty\omega}$-equivalence.

Invariants for $C^k_{\infty\omega}$-equivalence have been defined by Otto (see [17, 40, 41]) and been used to prove a number of results on the structure of fixed point logic with counting, on the relationship of (FP + C) with other logics and on the canonization problem with respect to $C^2_{\infty\omega}$-equivalence.

We give an informal description of $C^k_{\infty\omega}$-invariants. For k-tuples $\bar{a}, \bar{a}'$ from a fixed structure $\mathfrak{A}$, we write $\bar{a} \sim \bar{a}'$ to denote that $(\mathfrak{A}, \bar{a})$ and $(\mathfrak{A}, \bar{a}')$ are $C^k_{\infty\omega}$-equivalent. We write $[\bar{a}]$ for the $\sim$-equivalence class of $\bar{a}$, also called the $C^k_{\infty\omega}$-type of $\bar{a}$.

The desired $C^k_{\infty\omega}$-invariant of a structure $\mathfrak{A}$ has the form

$$I^k(\mathfrak{A}) = (\mathfrak{B}, v_1, \ldots, v_k)$$

where $\mathfrak{B} = (A^k/\sim, \prec, \ldots)$ is an *ordered* structure over the set of $C^k_{\infty\omega}$-equivalence classes in A^k, and where weight functions $v_j : (A^k/\sim) \longrightarrow \mathbb{N}$ associate with every type $[\bar{a}]$ the number

$$v_j([\bar{a}]) := \#\{b \in A : \bar{a} \sim \bar{a}\tfrac{b}{j}\}.$$

With the game characterization of $C^k_{\infty\omega}$-equivalence it can be shown that both $\sim$ and a total order $\prec$ on $A^k/\sim$ (which is a pre-order on A^k) can be inductively defined. One starts with an arbitrary ordering $\prec_0$ of the atomic types in k variables. At every stage a pre-order $\prec_i$ on A^k is defined such that the associated equivalence relation $\sim_i$ (i.e. $\bar{a} \sim_i \bar{a}'$ iff neither $\bar{a} \prec_i \bar{a}'$ nor $\bar{a}' \prec_i \bar{a}$) describes that Player II can maintain her winning condition for at least i moves. The refinement step can be derived from Theorem 5.3: $\bar{a} \prec_{i+1} \bar{a}'$ if either $\bar{a} \prec_i \bar{a}'$, or $\bar{a} \sim_i \bar{a}'$ and the following condition holds:

For the sequence $C_1 \prec_i C_2 \prec_i \cdots \prec_i C_r$ of $\sim_i$-equivalence classes, there exist $m \leq r$ and $j \leq k$ such that $\#\{b \in A : \bar{a}\frac{b}{j} \in C_m\} < \#\{b \in A : \bar{a}'\frac{b}{j} \in C_m\}$ and for all pairs $(\ell, i) <_{\text{lex}} (m, j)$ we have that $\#\{b \in A : \bar{a}\frac{b}{i} \in C_\ell\} = \#\{b \in A : \bar{a}'\frac{b}{i} \in C_\ell\}$.

Note that this refinement process is a variant of the colour refinement method leading to the stable colouring of a graph (see Example 4.23).

It follows from this description that the limits $\prec$ and $\sim$ of this inductive process are definable in (FP + C). In fact, a weaker logic is sufficient, namely fixed point logic together with a simple form of *cardinality comparison* which is captured by the so-called Rescher quantifier.

Definition 5.11 The *Rescher quantifier* is a Lindström quantifier which combines two given formulae together, binding a single variable in each of the two formulae. From $\psi(x, \bar{z})$ and $\varphi(y, \bar{z})$ the new formula [Resch $xy\ \psi(x, \bar{z}), \varphi(y, \bar{z})$] is formed. Its semantic is defined by the equivalence

$$\models [\text{Resch } xy\ \psi(x, \bar{z}), \varphi(y, \bar{z})] \longleftrightarrow \Big(\#_x[\psi(x, \bar{z})] < \#_y[\varphi(y, \bar{z})]\Big).$$

We write FP[Resch] for the logic obtained by adjoining the Rescher quantifier to FP.

Besides the relation $\sim$ (for equality), $\prec$ (for the linear order), and the already described weight functions $v_1, \ldots, v_k$, the structure $I^k(\mathfrak{A})$ is endowed with some additional relations to make sure that it encodes the entire $C^k_{\infty\omega}$-theory of $\mathfrak{A}$.

Atomic types: For every atomic type $t(x_1, \ldots, x_k)$ of vocabulary Υ_a, $I^k(\mathfrak{A})$ contains a unary relation $P_t := \{[\bar{a}] \in A^k/\sim : \mathfrak{A} \models t(\bar{a})\}$.

Reachability relations: For $j = 1, \ldots, k$, $I^k(\mathfrak{A})$ contains a binary relation $E_j([\bar{a}][\bar{a}'])$ which indicates that the type $[\bar{a}']$ can be obtained from $[\bar{a}]$, by changing the j-th coordinate. In other words,

$$E_j := \{([\bar{a}][\bar{a}']) : (\exists b \in A)\ \bar{a}\tfrac{b}{j} \sim \bar{a}'\} = \{([\bar{a}][\bar{a}\tfrac{b}{j}]) : b \in A\}.$$

Permutations: For every permutation $\sigma \in S_k$, we incorporate a binary relation $T_\sigma := \{([\bar{a}][\bar{a}']) : \sigma(\bar{a}) \sim \bar{a}'\}$ where $\sigma(a_1, \ldots, a_k) := a_{\sigma(1)}, \ldots, a_{\sigma(k)}$.

Obviously these additional relations are easily definable from $\mathfrak{A}$ and $\sim$. Further, it should be noted that there is some redundancy in this description in the sense that some relations are definable from others.

We can summarize the result on $C^k_{\infty\omega}$-invariants as follows.

Theorem 5.12 [17, 40, 41] *For every k and every finite relational vocabulary Υ_a there exists a function I^k, associating with every structure $\mathfrak{A} \in \mathrm{Fin}(\Upsilon_a)$ the $C^k_{\infty\omega}$-invariant $I^k(\mathfrak{A}) = (\mathfrak{B}, v_1, \ldots, v_k)$ such that the following hold:*

(i) The mapping $\mathfrak{A} \mapsto \mathfrak{B}$ is definable in FP[Resch].

(ii) For every $j \leq k$, the weight function $v_j : (A^k/\sim) \longrightarrow \mathbb{N}$ is definable from $\mathfrak{A}$ by a counting term $v(\bar{x}) = \#_y[\varphi(\bar{x}, y]$ with $\varphi \in$ FP[Resch].
(iii) $\mathfrak{A}$ and $\mathfrak{B}$ are $C^k_{\infty\omega}$-equivalent if and only $I^k(\mathfrak{A}) \cong I^k(\mathfrak{B})$.

Corollary 5.13 [17, 40] *For every class $\mathcal{K} \subseteq$ Fin(Υ_a), the following are equivalent*

(i) $\mathcal{K}$ is definable in (FP + C).
(ii) For some $k \in \mathbb{N}$, $\{I^k(\mathfrak{A}) : \mathfrak{A} \in \mathcal{K}\}$ is decidable in polynomial time.

Since the distinguishing power of the infinitary term calculus $T^k_{\infty\omega}$ can be reduced to $C^k_{\infty\omega}$-inequivalence of the corresponding finite structures, we obtain a notion of $T^k_{\infty\omega}$-invariants. It turns out that the $T^k_{\infty\omega}$-invariant $J^k(\mathfrak{D})$ of an arithmetical structure $\mathfrak{D}$ can be represented by a single natural number, and that J^k actually is an FP*-definable global function.

Theorem 5.14 *For every k and every vocabulary Υ of arithmetical structures there exists a numerical invariant $J^k : M_\Upsilon[\mathfrak{N}] \rightarrow \mathbb{N}$ with the following properties*

(i) J^k is FP**-definable.*
(ii) For all $\mathfrak{D}, \mathfrak{D}' \in M_\Upsilon[\mathfrak{N}]$

$$\mathfrak{D} \equiv_{T^k_{\infty\omega}} \mathfrak{D}' \ \textit{ iff } \ J^k(\mathfrak{D}) = J^k(\mathfrak{D}').$$

We sketch the proof. From Theorem 5.10 we know that $\mathfrak{D}$ and $\mathfrak{D}'$ are $T^k_{\infty\omega}$-equivalent, if and only if the corresponding finite structures fin$(\mathfrak{D})$ and fin$(\mathfrak{D}')$ are $C^k_{\infty\omega}$-equivalent. We cannot directly use the invariant $I^k(\text{fin}(\mathfrak{D}))$, due to the infinite vocabulary of fin$(\mathfrak{D})$. However, an inductive process, similar the the one defined above, can be used to work directly with $\mathfrak{D}$, rather than with fin$(\mathfrak{D})$. It is obvious that FP[Resch] and the simple applications of counting needed for defining the weight functions can be simulated in FP* with secondary part $\mathfrak{N}$. Further an ordering (or pre-ordering) on the primary part induces a ranking (or pre-ranking) of points: just assign to a point the number of smaller points. We thus obtain an FP* definable function, mapping every $\mathfrak{D} \in M_\Upsilon[\mathfrak{N}]$ to a ranked arithmetical structure that characterizes $\mathfrak{D}$ up to $T^k_{\infty\omega}$-equivalence. Finally we can use the same techniques as in the proof of the Coding Lemma in the previous section to encode this structure by a natural number.

With these invariants, we easily get a converse for Proposition 4.25 for the case that $\mathfrak{N}$ = PTA.

Theorem 5.15 *A class $\mathcal{K} \subseteq$ Fin(Υ_a) is* FP**-definable over PTA, if and only if $\mathcal{K}$ is* (FP + C)*-definable.*

Proof. The only-if direction has already been established. Suppose $\mathcal{K}$ is (FP + C)-definable. This and the FP*-definability of J^k imply that the class $\{J^k(\mathfrak{A}_\mathfrak{N}) : \mathfrak{A} \in \mathcal{K}\} \subseteq \mathbb{N}$ is decidable in polynomial-time and therefore expressible by a basic PTA-predicate. Since J^k is FP*-definable, the result follows. □

6 Asymptotic probabilities

Among the most beautiful results in finite model theory are the limit laws (in particular 0-1 laws) for various logics and probability distributions (see [11] for a survey).

We consider similar questions for metafinite structures, with fixed secondary part. It turns out, that limit laws hold only in rather restricted cases. Nevertheless, it is interesting to investigate and classify these cases.

Probability distributions. Fix a vocabulary $\Upsilon = (\Upsilon_a, \Upsilon_r, \Upsilon_w)$ where Υ_a and Υ_w are finite. Furthermore, fix a Υ_r-structure $\mathfrak{R}$, together with a probability distribution ν on the universe R. Finally, fix for every $n \in \mathbb{N}$ a probability distribution, over the finite set of Υ_a-structures with universe $\mathbf{n} = \{0, \ldots, n-1\}$. In this paper, μ_n will always be the uniform distribution, giving equal probability to all structures.

We define, for every $n \in \mathbb{N}$, a measure λ_n on the space S_n of metafinite structures $\mathfrak{D} \in M_\Upsilon[\mathfrak{R}]$ whose primary part has universe $\mathbf{n}$. The measure is defined by the following experiment:

- The primary part $\mathfrak{A}$ of $\mathfrak{D}$ is chosen according to the distribution μ_n.
- For every function symbol $w \in \Upsilon_w$ and every tuple $\bar{a}$, the value $w^{\mathfrak{D}}(\bar{a})$ is selected according to distribution ν.

Thus the measure λ_n defined in this way on S_n is the product measure of the uniform distribution μ_n over the finite set of primary parts with the product of $\sum_{w \in \Upsilon_w} n^{\text{arity}(w)}$ copies of ν. We denote the sequence $\lambda_1, \lambda_2, \ldots$ by $\boldsymbol{\lambda}$. For any class $C \subseteq M_\Upsilon[\mathfrak{R}]$ of metafinite structures we let

$$\lambda_n(C) := \lambda_n(C \cap S_n).$$

We now can define the corresponding probabilities of a sentence φ in any logic L of metafinite structures as follows:

$$\lambda_n(\varphi) = \lambda_n(\{\mathfrak{D} \in S_n : \mathfrak{D} \models \varphi)\}.$$

If the limit $\boldsymbol{\lambda}(\varphi) = \lim_{n\to\infty} \lambda_n(\varphi)$ exists, we call it the *asymptotic probability of* φ. If this limit exists for every sentence of L, then we say that the *convergence law* holds for L with respect to $\boldsymbol{\lambda}$. If, in addition, every sentence has asymptotic probability either 0 or 1, we say that the *0-1 law* holds for L with respect to $\boldsymbol{\lambda}$.

There are also other, weaker notions of limit laws, such as the existence of *Cesaro limits*

$$\lim_{n\to\infty} (\lambda_1(\varphi) + \lambda_2(\varphi) + \cdots + \lambda_n(\varphi))/n$$

or the *weak convergence law* (introduced by Shelah who called it the very weak 0-1 law [44]), saying that

$$\lim_{n\to\infty} \lambda_{n+1}(\varphi) - \lambda_n(\varphi) = 0.$$

It is clear that already very little of arithmetic present in $\mathfrak{R}$ suffices to refute the convergence law. If $\mathfrak{R}$ contains the natural numbers and parity is definable, and if we have summation over multisets then we can say that the number of elements of $\mathfrak{A}$ is even. This holds even for the trivial situation that $\Upsilon_a = \Upsilon_w = \varnothing$.

Thus, the question whether a convergence law or a 0-1 law holds, is interesting only for rather limited secondary parts $\mathfrak{R}$. In the sequel, we consider classes of *simple metafinite structures*, with various cases of $\mathfrak{R}, \Upsilon$ and ν.

6.1 The uncountable case

It should be noted that $\lambda_n(\varphi)$ is not defined in all situations. In fact, if φ is infinitary, then the set $\{\mathfrak{D} \in S_n : \mathfrak{D} \models \varphi\}$ need not be measurable. We show this by means of an example (that uses the axiom of choice and the continuum hypothesis).

Proposition 6.1 *Let* $\mathfrak{R} = (R, 0, 1, \frac{1}{2}, +, \cdot, \leq)$, *where* R *is the real interval* $[0,1]$ *and* $+$ *is addition modulo 1. Let* $\Upsilon_a = \varnothing, \Upsilon_w = \{c\}$ *where* c *is nullary. Then, even for* $n = 1$, *there is no probability distribution on* $[0,1]$ *under which every sentence of* $L^0_{\infty\omega}$ *defines a measurable subset of* S_n *and every singleton has probability 0.*

Proof. It is known that, on the basis of the axiom of choice and the continuum hypothesis, there exists no probability distribution on $[0,1]$, giving probability 0 to singletons, such that all subsets of $[0,1]$ are measurable.

It therefore suffices to show that for every set $X \subseteq [0,1]$ there exists a sentence $\psi_X \in L^\omega_{\infty\omega}$ such that for structure $\mathfrak{D} \in M_\Upsilon[\mathfrak{R}]$

$$\mathfrak{D} \models \psi_X \text{ if and only if } c^{\mathfrak{D}} \in X.$$

Every real number $r \in [0,1]$ can be approximated by sequences $(a_n)_{n\in\omega}$ and $(b_n)_{n\in\omega}$ of dyadic rational numbers (i.e. rationals whose denominators are powers of two) such that $a_n \leq r \leq b_n$ and

$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = r.$$

Every dyadic rational in $[0,1]$ is representable by a basic weight term in our language. Thus, in $L^\omega_{\infty\omega}$, we can form the sentence

$$\varphi_r := \bigwedge_{n\in\omega} (a_n \leq c \wedge c \leq b_n)$$

expressing that that $c = r$. Now the sentence $\psi_X := \bigvee_{r\in X} \varphi_r$ asserts that $c \in X$, which is what we wanted to prove. □

Even though atomic formulae over $\mathfrak{R}$ define very simple sets, measurability need not be preserved under unrestricted conjunctions and disjunctions available in $L^0_{\infty\omega}$. Fortunately, there exist reasonable conditions on a logic L and a secondary part $\mathfrak{R}$ such that all L-definable model subclasses in S_n are measurable.

Definition 6.2 Let $\mathfrak{R}$ be a structure over Υ_r and ν a probability distribution on R. We say that $\mathfrak{R}$ has measurable atoms with respect to ν if every (first-order) atomic formula $\varphi(z_1, \ldots, z_t)$ of vocabulary Υ_r defines a measurable set so that $\nu(\{\bar{u} \in R^t : \mathfrak{R} \models \varphi(\bar{u})\})$ is defined.

Proposition 6.3 *If $\mathfrak{R}$ has measurable atoms with respect to ν, and every L-formula is equivalent to a formula in $L^{\omega}_{\omega_1\omega}$, then, for every n, every L-definable model class in S_n is measurable with respect to λ_n.*

Proof. Fix a primary part $\mathfrak{A}$ with universe $\mathbf{n}$ and let $S(\mathfrak{A})$ be the set of structures $\mathfrak{D} \in S_n$ with primary part $\mathfrak{A}$. Since, for fixed n, there are only finitely many primary parts, it suffices to show that the set $\{\mathfrak{D} \in S(\mathfrak{A}) : \mathfrak{D} \models \psi\}$ is measurable for every fixed $\mathfrak{A}$ and every sentence $\psi \in L^{\omega}_{\omega_1\omega}$. Then $\{\mathfrak{D} \in S_n : \mathfrak{D} \models \psi\}$ is a finite union of measurable sets and thus measurable.

It suffices to prove the claim for the expansion of the structure $\mathfrak{A}$ with names for all elements of A. We therefore suppose without loss of generality, that every element of $\mathfrak{A}$ is an individual constant. On $S(\mathfrak{A})$, the logic $L^{\omega}_{\omega_1\omega}$ then admits the elimination of quantifiers and of all primary relation and function symbols, except the constants: Every quantifier $\exists x\beta$ is replaced by $\bigvee_{a\in A} \beta(a/x)$, every primary term by the name of its value and every primary atomic subformula $Q(\bar{a})$ by its truth value. Thus the given sentence ψ is equivalent to a quantifier-free sentence φ. Since weight terms $w(\bar{a})$ are random variables with respect to the distribution ν and since $\mathfrak{R}$ has measurable atoms with respect to ν, it follows that for every atomic formula $\alpha = P(w_1(\bar{a}_1), \ldots, w_k(\bar{a}_k))$ that may occur in φ, the set $\{\mathfrak{D} : \mathfrak{D} \in S(\mathfrak{A}) \wedge \mathfrak{D} \models \alpha\}$ is measurable. Since the measurable sets are closed under complementation and under countable unions and intersections the claim follows. □

Examples. We now consider some specific examples for $\mathfrak{R}$, ν, Υ_a and Υ_w such that the existence of a convergence law or a 0-1 law for first-order logic can be easily reduced to known results in finite model theory. We write FO for first-order logic in the classical sense, and FO* for its extension to first-order logic of metafinite structures.

One unary weight function into an uncountable linear order. Let $\mathfrak{R} = ([0,1], <)$ with the uniform measure on $[0,1]$, let Υ_a be an arbitrary finite relational vocabulary and $\Upsilon_w = \{w\}$ with w unary.

For any metafinite structure $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, \{w\}) \subseteq M_\Upsilon[\mathfrak{R}]$, the weight function w defines a partial order on A by

$$a < b \text{ iff } \mathfrak{D} \models w(a) < w(b).$$

If $\mathfrak{D}$ is chosen randomly, then almost surely $\mathfrak{D} \models \forall x \forall y\ w(x) \neq w(y)$, so $<$ is in fact a random total order on $\mathfrak{A}$. Replacing $w(x)$ by x we can translate every sentence $\psi \in \mathrm{FO}^*$ to a sentence $\varphi \in \mathrm{FO}$ such that, almost surely, $\mathfrak{D} \models \psi$ if and only if $(\mathfrak{A}, <) \models \varphi$.

The problem is thus reduced to a problem on a class of *random finite ordered structures.*

For specific results, we distinguish several cases according to the vocabulary Υ_a of the primary part:

$\Upsilon_a = \emptyset$**:** In this case the structures have the form $\mathfrak{D} = (\mathbf{n}, \mathfrak{R}, \{w\})$ and the reduction gives a pure linear order $(\mathbf{n}, <)$. It is well-known that no first-order sentence φ can distinguish between linear orders $(\mathbf{n}, <)$ and $(\mathbf{m}, <)$ if both n and m are larger than a constant n_0 that depends only on the quantifier-rank of φ. Thus, we have a 0-1 law for FO.
However, in logics with recursion, such as transitive closure logic or fixed point logic, the presence of a linear order suffices to express that the structure has an even number of elements, and we therefore do not have any convergence law for these stronger logics. The same applies to monadic second-order logic MSO.

Υ_a **is monadic:** Clearly, we no longer have a 0-1 law. The sentence

$$\forall x([\forall y\ w(x) \leq w(y)] \rightarrow Px)$$

expresses, that the elements with minimal weights satisfy P. This is true with probability 1/2 in all cardinalities.
However, we still have the convergence law, because of the convergence law for the first-order logic of random monadic structures with a linear order. This results appears in [37] but is attributed there to Ehrenfeucht.

Υ_a **contains at least one binary predicate.** Here we have non-convergence, due to the result of Compton, Henson and Shelah [12], that on the class of random ordered graphs there exist first-order sentences without an asymptotic probability.

Two unary functions into an uncountable linear order. For structures $\mathfrak{D} = (\mathfrak{A}, \mathfrak{R}, \{v, w\})$ with two unary weight functions into $\mathfrak{R} = ([0,1], <)$, it is easy to see that we no longer have a 0-1 law. For instance, the sentence

$$\exists x \exists y (\forall z (v(x) \leq v(z) \wedge w(y) \leq w(z)) \wedge v(x) < w(y)),$$

expressing that the minimal v-weight is smaller than the minimal w-weight, is true with probability 1/2 in all cardinalities. In fact, we don't even have the convergence law. With two weight functions we can almost surely interpret two-dimensional partial orders (i.e. the intersection of two linear orders), and it is a result of Spencer [45], that there exist first-order sentences without asymptotic probabilities for k-dimensional partial orders, whenever $k \geq 2$.

Field of reals as secondary part. A different class of examples is obtained by for the secondary part the field of reals $\mathfrak{R} = (\mathbb{R}, +, \cdot, 0, 1)$.

Here we have a 0-1 law for arbitrary relational Υ_a and arbitrary Υ_w. This might come as a surprise, but it is true for rather trivial reasons: Take any pair of basic weight terms $F(\bar{x}), G(\bar{y})$. Then almost surely either $\mathfrak{D} \models \forall \bar{x} \forall \bar{y}\ F(\bar{x}) = G(\bar{y})$ or $\mathfrak{D} \models \forall \bar{x} \forall \bar{y}\ F(\bar{x}) \neq G(\bar{y})$. Thus, the secondary part almost surely provides no information at all, so the 0-1 law holds whenever it holds on finite structures.

6.2 The countable case

The other interesting case is when the secondary part is countable. We may assume that its universe is the set of natural numbers. Then ν is given by a sequence p_n of nonnegative reals such that $\sum_{n=0}^{\infty} p_n = 1$ and $p_n = \nu(\{n\})$. We first show that one gets a strong form of non-convergence even in very simple cases. As above, $\boldsymbol{\lambda} = \lambda_1, \lambda_1, \ldots$ is the sequence of distributions induced by $\boldsymbol{\nu}$.

Definition 6.4 A distribution $\boldsymbol{\nu}$ ***decreases rapidly*** if $\lim_{n\to\infty} \frac{p_{n+1}}{p_n} = 0$.

An example of a rapidly decreasing distribution is the *Poisson distribution* $p_n := e^{-\mu}\mu^n/n!$ with the mean value μ.

Proposition 6.5 *Suppose that $\Upsilon_a = \Upsilon_r = \varnothing$ and Υ_w consists of one unary function name w, and let $\boldsymbol{\lambda}$ be induced by a rapidly decreasing distribution ν. Then the sentence*

$$\varphi = \exists x \forall y (y \neq x \to w(x) \neq w(y)).$$

has no asymptotic probability with respect to $\boldsymbol{\lambda}$. Even the Cesaro probabilities

$$\chi_k(\varphi) = [(\lambda_1(\varphi) + \cdots + \lambda_k(\varphi)]/k$$

do not converge.

Proof. We start with preliminary observations. Since $p_{n+1}/p_n = 0$ tends to 0, for every $c < 1$, there exists $m = m(c)$ such that $p_{n+1}/p_n < c$ for all $n > m$. Thus we may assume without loss of generality that $p_{n+1} < p_n/4$ for all n.

The sum $\sum_{j\geq n} p_j \Big/ p_n = 1$ converges to 1 as n grows to infinity. Indeed, for every $\varepsilon > 0$, there exists a positive $c < 1$ such that $(c/(1-c)) < \varepsilon$. Let $m = m(c)$ be as above and suppose that $n > m$. We have

$$\sum_{j\geq n} \frac{p_j}{p_n} < \sum_{j\geq n} c^{j-n} = 1 + c/(1-c) < 1 + \varepsilon.$$

Finally, $e^{-2} < (1-p)^{1/p} < e^{-1}$ if $0 < p < 1/2$. Indeed, apply the Mean Value Theorem to the function $f(t) = -\log(1-t)$ on the interval $[0, p]$. There is a point $t \in (0, p)$ such

$$f(p) - f(0) = -\log(1-p) = (p-0)f'(t) = p/(1-t).$$

Since $p < p/(1-t) < p/(1-p) < p/(1-1/2) = 2p$, we have $p < -\log(1-p) < 2p$ and therefore $e^{-2p} < 1 - p < e^{-p}$. Now raise the terms to power $1/p$.

Now we are ready to prove the proposition. The idea is as follows. Let $p = p_i$ and $M = \lfloor 1/p \rfloor$, so that $Mp \to 1$ and M grows much faster than i. We will check that the probabilities $\lambda_M(\exists x! [w(x) = i])$ converge to a positive number and therefore the probabilities $\lambda_M(\varphi)$ have a positive limes inferior. Further, let $N = \lfloor 1/\sqrt{p_{i+1}p_i} \rfloor$, so that $Np_i \to \infty$ and $Np_{i+1} \to 0$. We will check that the probabilities $\lambda_N(\exists x [w(x) > i])$ converge to zero and the probabilities $\lambda_N(\bigvee_{j\leq i} \exists x! [w(x) = j])$ converge to zero. Therefore probabilities $\lambda_N(\varphi)$ converge to zero, because, for every n,

$$\lambda_n(\varphi) \leq \lambda_n(\bigvee_{j\leq i} \exists x![w(x) = j]) + \lambda_n(\exists x[w(x) > i]).$$

Now let us do the necessary computations.

Part 1. Let n range over the interval $[M, 2M]$ and $\epsilon(p) = (1-p)^{1/p} - e^{-1}$, so that $\epsilon(p) = o(1)$ as p tends to 0. We have

$$\begin{aligned}\lambda_n(\varphi) &\geq \lambda(\exists! x[w(x) = i]) \geq np(1-p)^{n-1} > np[(1-p)^{1/p}]^{np} \\ &= np[e^{-1} + \epsilon(p)]^{np} > Mp[e^{-1} + \epsilon(p)]^{2Mp} = e^{-2} + o(1).\end{aligned}$$

It follows that

$$\liminf_k \chi_k(\varphi) \geq \liminf \chi_{2M}(\varphi) \geq \frac{1}{2M}[0 + M(e^{-2} + o(1))] > \frac{1}{18}.$$

Part 2. Let n range over $[N+1, 18N]$. We have

$$\begin{aligned}\lambda_n(\exists x[w(x) > i]) &\leq 18N \sum_{j=i+1}^{\infty} p_j \\ &\leq 18 \frac{1}{\sqrt{p_i p_{i+1}}} \sum_{j=i+1}^{\infty} p_j \\ &= \frac{1}{\sqrt{p_i p_{i+1}}} p_{i+1}(1 + o(1)) \\ &= \sqrt{\frac{p_{i+1}}{p_i}}(1 + o(1)) = o(1).\end{aligned}$$

Further, let j range over natural numbers $\leq i$. We have

$$\begin{aligned}\lambda_n(\bigvee_j \exists x![w(x) = j]) &\leq \sum_j \lambda_n(\exists x![w(x) = j]) \\ &= \sum_j np_j(1-p_j)^{n-1} \\ &< \sum_j 18Np_j(1-p_j)^N \\ &= 18N \sum_j p_j \left[\left(1 - \frac{1}{p_j}\right)^{1/p_j}\right]^{Np_j} \\ &< 18N \sum_j p_j e^{-Np_j} \\ &\leq 18 \sum_j Np_j e^{-Np_j}.\end{aligned}$$

Notice that $Np_j - Np_{j+1} > 1$ if $j < i$. Indeed, $Np_j > Np_i \geq \sqrt{p_i/p_{i+1}} > 2$ because every $p_{m+1} < p_m/4$. Further, $Np_{j+1} < Np_j/4$. Hence $Np_j - Np_{j+1} > (3/4)Np_j > 1$. Therefore

$$\lambda_n(\bigvee_j \exists x![w(x) = j]) \leq 18 \cdot \sum_{m=\lfloor Np_i \rfloor}^{\infty} me^{-m},$$

which converges to 0 when i grows to infinity because the series $\sum_{m=0}^{\infty} me^{-m}$ is convergent and $Np_i \leq p_i/\sqrt{p_i p_{i+1}} = \sqrt{p_i/p_{i+1}} \to \infty$. Consequently, $\lambda_n(\varphi) = o(1)$ and therefore

$$\liminf_k(\chi_k(\varphi)) \leq \chi_{18N}(\varphi) \leq \frac{1}{18N}([\sum_{m=1}^{N} \lambda_n(\varphi)] + 17N\,o(1)) \leq 1/18.$$

□

However, there is a weaker form of limit law, introduced by Shelah, which is of interest for this case.

Definition 6.6 We say that a class of sentences L satisfies the *weak convergence law* with respect to $\boldsymbol{\lambda} = (\lambda_n)_{n \in \mathbb{N}}$ if for all $\psi \in L$ we have that

$$\lim_{n \to \infty} \lambda_{n+1}(\psi) - \lambda_n(\psi) = 0.$$

For instance, it has been proved by Shelah [44], that first-order logic satisfies the weak convergence law on ordered random graphs and also on a random binary function. We can prove a similar results for monadic classes of metafinite structures with an arbitrary countable secondary part.

Theorem 6.7 *Let $\mathfrak{N}$ be any structure with universe $\mathbb{N}$, endowed with an arbitrary probability distribution ν, and let Υ_a and Υ_w be unary. Then for the induced sequence $\boldsymbol{\lambda}$ of probability distributions, first-order logic satisfies the weak convergence law.*

Proof. Let $\mathfrak{D} = (\mathfrak{A}, \mathfrak{N}, W)$ and $\mathfrak{D}' = (\mathfrak{B}, \mathfrak{N}, W')$ be two structures in $M_\Upsilon[\mathfrak{N}]$. Recall that for $a \in A$ and $b \in B$,

$$(\mathfrak{D}, a) \sim_0 (\mathfrak{D}', b)$$

means that the function $p : a \mapsto b$ is a partial isomorphism from $\mathfrak{D}$ to $\mathfrak{D}'$, i.e. that a and b satisfy the same Υ_a-relations over $\mathfrak{A}$ and $\mathfrak{B}$, respectively, and that the weight functions map a and b to the same values of $\mathbb{N}$. For every $m \in \mathbb{N}$, we say that a $\sim_0$-equivalence class C is m-bounded, if $w^{\mathfrak{D}}(a) \leq m$ for all $w \in \Upsilon_w$ and $(\mathfrak{D}, a) \in C$.

The structures $\mathfrak{D}$ and $\mathfrak{D}'$ are k-equivalent, i.e. cannot be distinguished by formulae of quantifier depth k, if every $\sim_0$-equivalence class C contains the same number of elements in $\mathfrak{D}$ and $\mathfrak{D}'$, or more than k elements in both structures. This can be proved by a straightforward application of Ehrenfeucht-Fraïssè games.

For every $\varepsilon > 0$ take a large enough natural number m so that $\sum_{i=0}^{m} p_i \geq 1-\varepsilon$. Given k, choose n_0 large enough such that for every $n > n_0$, a random $\mathfrak{D} \in S_n$ contains, with probability at least $1 - \varepsilon$, more than k elements in every m-bounded $\sim_0$-equivalence class.

The process of drawing a random structure $\mathfrak{D} \in S_{n+1}$ can be described as follows: first we choose a random structure $\mathfrak{D}' \in S_n$; then we add a new element a and determine at random the truth values of atoms Pa for $P \in \Upsilon_a$ and the values of the weight terms $w(a)$ for $w \in \Upsilon_w$. With probability at least $(1-\varepsilon)^{\ell}$ (where $\ell = |\Upsilon_w|$), the $\sim_0$-equivalence class of a is m-bounded. As a consequence, if $n > n_0$, then $\mathfrak{D}$ and $\mathfrak{D}'$ differ by an element that almost surely belongs to a class with more than k representants in both structures. Thus, $\mathfrak{D}$ is almost surely k-equivalent to $\mathfrak{D}'$.

Since k was arbitrary, it follows that that for every first-order formula ψ

$$\lim_{n\to\infty} \lambda_n(\psi) - \lambda_{n+1}(\psi) = 0.$$

□

Remark. With the same argument, the weak convergence law also holds for $L^{\omega}_{\omega_1\omega}$.

Acknowledgements

We would like to thank David Harel, Martin Otto, Jurek Tyszkiewicz, Moshe Vardi and Victor Vianu for valuable comments and suggestions.

The results in Sect. 5 were strongly influenced by discussions with Martin Otto. Some of the results in Sect. 6 were contributed by or proved in collaboration with Jurek Tyszkiewicz [46].

References

1. S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases*, Addison Wesley (1994).
2. S. Abiteboul and V. Vianu, *Generic Computation and Its Complexity*, Proceedings of 23rd ACM Symposium on Theory of Computing (1991), 209–219.
3. L. Adleman and K. Manders, *Computational complexity of decision problems for polynomials*, Proceedings of 16th IEEE Symposium on Foundations of Computer Science (1975), 169–177.
4. L. Adleman and K. Manders, *Diophantine Complexity*, Proceedings of 17th IEEE Symposium on Foundations of Computer Science (1976), 81–88.
5. S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, *Proof verification and intractability of approximation problems*, Proceedings of 33rd IEEE Symposium on Foundations of Computer Science (1992), 210–214.
6. J. Barwise, *On Moschovakis closure ordinals*, Journal of Symbolic Logic **42** (1977), 292–296.
7. L. Blum, M. Shub, S. Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*, Bull. Amer. Math. Soc.**21** (1989), 1–46.

8. E. Börger, *Annotated Bibliography on Evolving Algebras*, in: E. Börger (Ed.), Specification and Validation Methods, Oxford University Press, to appear.
9. J. Cai, M. Fürer and N. Immerman, *An Optimal Lower Bound on the Number of Variables for Graph Identification*, Proceedings of 30th IEEE Symposium on Foundations of Computer Science (1989), 612–617.
10. A. Chandra and D. Harel, *Computable Queries for Relational Data Bases*, Journal of Computer and System Sciences **21** (1980), 156–178.
11. K. Compton, *0-1 Laws in Logic and Combinatorics*, in: NATO Adv. Study Inst. on Algorithms and Order, I. Rival (Ed.), 1988, 353–383.
12. K. Compton, C. Henson and S. Shelah, *Nonconvergence, undecidability and intractability in asymptotic problems*, Annals of Pure and Applied Logic **36** (1987), 207–224.
13. P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, preprint (1995).
14. A. Dawar, *Feasible Computation through Model Theory*, PhD thesis, University of Pennsylvania (1993).
15. R. Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, SIAM-AMS Proceedings **7** (1974), 43–73.
16. E. Grädel and K. Meer, *Descriptive Complexity Theory over the Real Numbers*, Proceedings of 27th ACM Sympposium on Theory of Computing (1995).
17. E. Grädel and M. Otto, *Inductive Definability with Counting on Finite Structures*, in: Selected Papers, 6th Workshop on Computer Science Logic CSL 92, San Miniato 1992, Lecture Notes in Computer Science Nr. 702, Springer (1993), 231–247.
18. J. Gross and T. Tucker, *Topological Graph Theory*, Wiley, New York (1987).
19. S. Grumbach and J. Su, *Finitely representable databases*, Proceedings of 13th ACM Symposium on Principles of Database Systems (1994).
20. Y. Gurevich, *Toward logic tailord for computational complexity*, in: M. M. Richter et al. (Eds), Computation and Proof Theory, Springer Lecture Notes in Mathematics Nr. 1104 (1984), 175–216.
21. Y. Gurevich, *Logic and the Challenge of Computer Science*, in: E. Börger (Ed), Trends in Theoretical Computer Science, Computer Science Press (1988), 1–57.
22. Y. Gurevich, *Evolving Algebras 1993: Lipari Guide*, in: E. Börger (Ed.), Specification and Validation Methods, Oxford University Press, to appear.
23. Y. Gurevich and S. Shelah, *Fixed Point Extensions of First Order Logic*, Annals of Pure and Applied Logic **32** (1986), 265–280.
24. T. Hirst and D. Harel, *Completeness Results for Recursive Databases*, Journal of Computer and System Sciences, to appear. (Also: 12th ACM Symp. on Principles of Database Systems (1993), 244–252.)
25. T. Hirst and D. Harel, *More about Recursive Structures: Zero-One Laws and Expressibility vs. Complexity*, unpublished (1995).
26. B. Hodgson and C. Kent, *A Normal form for Arithmetical Representation of NP-sets*, Journal of Computer and System Sciences **27** (1983), 378–388.
27. N. Immerman, *Upper and lower bounds for first-order expressibility*, Journal of Computer and Systems Sciences **25** (1982), 86–104.
28. N. Immerman, *Relational Queries Computable in Polynomial Time*, Information and Control **68** (1986), 86–104.
29. N. Immerman, *Expressibility as a Complexity Measure: Results and Directions*, Proc. of 2nd Conf. on Structure in Complexity Theory (1987), 194–202.

30. N. Immerman, *Descriptive and Computational Complexity,* in: J. Hartmanis (Ed.), Computational Complexity Theory, Proc. of AMS Symposia in Appl. Math. **38** (1989), 75–91.
31. N. Immerman and E. Lander, *Describing Graphs: A First Order Approach to Graph Canonization,* in: A. Selman (Ed), Complexity Theory Retrospective. (In Honor of Juris Hartmanis), Springer, New York 1990, 59–81.
32. J. Jones and Y. Matijasevich, *Register machine proof of the theorem of exponential diophantine representation of enumerable sets*, Journal of Symbolic Logic 49 (1984), 818–829.
33. F. Kabanza, J. Stevenne and P. Wolper, *Handling Infinite Temporal Data,* to appear in Journal of Computer and System Sciences.A preliminary version appeared in Proceedings of 9th ACM Syposium on Principles of Database Systems (1990).
34. P. Kanellakis, *Elements of Relational Database Theory,* in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, vol. B, North Holland, Amsterdam 1990, pp. 1073–1156.
35. P. Kanellakis, G. Kuper and P. Revesz, *Constraint Query Languages,* Proceedings of 9th ACM Symposium on Principles of Database Systems (1990), 299–313.
36. C. Kent and B. Hodgson, *An arithmetical characterization of NP*, Theoretical Computer Science **12** (1982), 255–267.
37. J. Lynch, *Almost sure theories*, Annals of Mathematical Logic **18** (1980), 91–135.
38. Y. Matijasevich, *Hilbert's Tenth Problem*, MIT Press, Cambridge (1993).
39. M. Otto, *Generalized Quantifiers for Simple Properties,* Proceedings of IEEE Symposium on Logic in Computer Science (1994), 30–39.
40. M. Otto, *The Expressive Power of Fixed-Point Logic with Counting*, Journal of Symbolic Logic, to appear.
41. M. Otto, Habilitationsschrift, RWTH Aachen 1995.
42. C. Papadimitriou and M. Yannakakis, *Optimization, approximization and complexity classes*, Journal of Computer and System Sciences **43** (1991), 425–440.
43. B. Poizat, *Deux ou trois choses que je sais de L_n*, Journal of Symbolic Logic 47 (1982), 641–658.
44. S. Shelah, *The very weak zero-one law for random graphs with order and random binary functions*, preprint (1994).
45. J. Spencer, *Nonconvergence in the theory of random orders*, Order **7** (1991), 341–348.
46. J. Tyszkiewicz, private communication.
47. J. D. Ullman, *Database and Knowledge-Base Systems, Vol. I and II,* Computer Science Press (1989).
48. M. Vardi, *Complexity of Relational Query Languages,* Proc. of 14th Symposium on Theory of Computing (1982), 137–146.

Automatic Presentations of Structures

Bakhadyr Khoussainov and Anil Nerode

Introduction

In this paper we introduce the systematic study of presentations of algebraic structures by finite automata. We call these automatic structures.

The study of recursive presentations of algebraic structures was initiated by Froehlich and Shepherdson [8], Rabin [14], and Mal'cev [11]. Since then, recursive algebra has been an active area of study by researchers associated with Nerode in the U. S. and Ershov in Russia. A **recursive structure** is a countable structure equipped with Turing machines for deciding equality and the other atomic relations. The ordering of rationals, vector spaces over rational numbers, absolutely free algebras, free groups, and finitely presented algebras with decidable word problems have obvious recursive presentations. Recursive Boolean algebras, linear orderings, abelian groups, vector spaces, fields, lattices, and other structures have been studied extensively.

In the late 1980's Nerode, Remmel, and Cenzer [15] [16] developed a corresponding theory of p-time structures. These are recursive structures which are presented by polynomial time recursive functions. This has also become an active area of research. For example, Remmel proved that any recursive, purely relational, structure is recursively isomorphic to a p-time structure, and, as a corollary, that any recursive Boolean algebra has a polynomial time presentation.

In this paper we further restrict the recursive functions of the presentaions and insist they be given by finite automata. We obtain a very fine grain theory of automatic, or **automaton presentable** structures. That is, these are structures provided with finite automata for deciding equality on the domain and the atomic relations of the structure.

We note that automatic groups in a closely related sense are an active object of study ([6]) growing from the need to have feasible calculations in 3-manifold theory. Epstein, Cannon, Thurston have developed the theory of automatic groups motivated by performing computations on groups associated with 3-manifolds. They consider finitely generated groups with generators $g_1, \ldots, g_n$. Each generator g_i naturally defines a unary operation f_i on the domain of G by the right multiplication, that is $f_i(x) = xg_i$, where $x \in G$. Thus, with the finitely generated group G one can associate the unary structure $(G, f_1, \ldots, f_n)$. They call the group G automatic ([6]) if the corresponding unary structure $(G, f_1, \ldots, f_n)$ is automatic in our sense below. They do not impose the requirement that the binary group operation be described by a finite automaton.

They prove, for instance, that any automatic group is finitely presented [6]. Automatic groups in the sense of Epstein, Cannon, and Thurston, regarded as unary structures as above, are automatic in our sense.

Research on automatic structures, unlike research on recursive and p-time structures, concentrates on positive results. The results of Epstein,Cannon, Thurston show the usefulness of automaton presentations. If one has an automaton presentation of a structure $\mathcal{A}$, one can perform automaton computations on the structure. As an example, let P_S be the following problem: is there a recursive procedure which, when applied to a first order definition of a relation S on $\mathcal{A}$, yields an algorithm for deciding P. To solve this problem, we want to find presentations of $\mathcal{A}$ in which computations of atomic relations are governed by finite automata. In such a presentation, these computations can be performed in real time. If we find such a presentation, we can transform the problem P_S into a problem about finite automata. Since finite automata possess many decidability properties, we can deduce the decidability of many problems such as P_S. This, in turn, leads to consideration of the complexity of problems about automatic structures.

Here are several basic questions.

- What is an automatic presentation?
- Which structures are automaton presentable?
- What does automaton presentablity say about the structure?
- What are the easiest standard structures that have or lack automaton presentations?
- What is the complexity of problems formulated over automatic structures?

We define the notions of strongly automatic, automatic, and asynchronously automatic presentations. We provide many examples of automatic presentations of linear orderings, vector spaces, abelian groups, permutation structures, etc. Though these examples are not startling, they show that automatic presentations play a basic role underlying many recursive and polynomial time structures. We obtain an algebraic characterization of automatic structures, by introducing many sorted finite automata. This leads to a natural generalization of the Myhill-Nerode theorem charcterizing finite automaton recognizable languages by congruences of finite index. We also characterize structures which possess strongly automatic presentations. At the end we discuss automatic isomorphism types.

1 Preliminary Definitions

Consider structures of the form $(A, f_0^{n_0}, \ldots, f_k^{n_k}, P_0^{m_0}, \ldots, P_s^{m_s}, c_0, \ldots, c_t)$, where A is the **domain**, each $f_i^{n_i}$ is an **operation** of arity n_i on A, each $P_j^{m_j}$ is a **predicate** of arity m_j on A, and each **constant** c_i belongs to A. We suppose that the domain A is at most a countable set. The sequence

$$(f_0^{n_0}, \ldots, f_k^{n_k}, P_0^{m_0}, \ldots, P_s^{m_s}, c_0, \ldots, c_t)$$

is called the **signature** of the structure $\mathcal{A}$. Given a structure $\mathcal{A}$ of signature, form a new structure

$$\mathcal{A}_R = (A, F_0^{n_0+1}, \ldots, F_k^{n_k+1}, P_0^{m_0}, \ldots, P_s^{m_s}, c_0, \ldots, c_t),$$

where for all $a_1, \ldots, a_{n_i+1} \in A$, $F_i^{n_i+1}(a_1, \ldots, a_{n_i+1})$ iff $f_i^{n_i+1}(a_1, \ldots a_{n_i}) = a_{n_i+1}$. This makes the structure $\mathcal{A}_R$ **relational**. We often identify the structure $\mathcal{A}$ with $\mathcal{A}_R$.

Let Σ be a finite alphabet. Form the set $\Sigma^\star$ of all finite words of the alphabet Σ. A **(nondeterministic) finite automaton** over the alphabet Σ is a quadruple $\Omega = (S, I, \Delta, F)$, where S is the finite non-empty **set of states**, I is a non-empty subset of S, called the set of **initial states**, $\Delta \subseteq S \times \Sigma \times S$ is a non-empty set, called **the transition table**, F is a subset of S, called the set of **final states**. Thus, Δ can be viewed as a mapping from $S \times \Sigma$ to $P(S) = \{S' \mid S' \subset S\}$. The mapping Δ can be extended naturally to a mapping from $S \times \Sigma^\star$ to $P(S)$. If there is no confusion, denote this extension by Δ too. For convenience we present a finite state automaton Ω as a directed graph. The nodes of the graph are the states of the automaton Ω. The edge relation E on the nodes is defined as follows. There exists an edge connecting node s with node s' if and only if $(s, \sigma, s') \in \Delta$ for some $\sigma \in \Sigma$. We label this edge σ. Thus, $E = \{(s, s') \mid \exists \sigma \in \Sigma((s, \sigma, s') \in \Delta)\}$. The automaton Ω **accepts** $\sigma_1 \ldots \sigma_n$ if and only if there exists a path $s_0 \ldots s_n$ in the graph presenting the automaton such that $s_0 \in S_0$, $s_n \in F$, and for each i $(s_i, \sigma_i, s_{i+1}) \in \Delta$. We call such a path a **computation** of Ω on input $\sigma_1 \ldots \sigma_n$. The **behavior** of the automaton Ω is defined as the set $L(\Omega)$ of all words accepted by Ω. A set, or equivalently language, $D \subseteq \Sigma^\star$ is **finite automaton (FA) recognizable**, or simply **recognizable**, if there exists a finite automaton Ω such that $D = L(\Omega)$. It is well known that the set of all finite automata recognizable sets forms a Boolean algebra and that the emptiness problem for finite automata is decidable.

Dealing with an automatic structure $\mathcal{A}$, it is necessary to have an automaton that recognizes the domain of the structure. To do this we have to **present** elements of the structure as words of a finite alphabet. Thus, suppose that a structure $\mathcal{A}$ is given. An **automatic presentation of the domain** of $\mathcal{A}$ is a surjective mapping $\nu : D \to A$, where D is a recognizable subset of $\Sigma^\star$. If $a \in A$ and $\nu(\alpha) = a$, we say that α **presents** the element a. Having the mapping ν, we formulate three problems.

1. Find a procedure which, for any two words α, $\beta \in D$ decides whether $\nu(\alpha) = \nu(\beta)$.

2. Find a procedure which, for each predicate $P_j^{m_j}$, and all $\alpha_1, \ldots, \alpha_{m_j} \in D$, decides whether $P_j^{m_j}(\nu(\alpha_1), \ldots, \nu(\alpha_{m_j}))$ holds.

3. Find a procedure which for each operation $f_i^{n_i}$, and all $\alpha_1, \ldots, \alpha_{n_i+1} \in D$ decides whether $F_i^{n_i+1}(\nu(\alpha_1), \ldots, \nu(\alpha_{n_i+1}))$ holds.

Informally, a presentation ν is an **automatic presentation** of $\mathcal{A}$ if there exist automata for deciding the above three problems. This considerations suggest the definition of automata recognizable relations over $\Sigma^\star$.

1. n-variable Strong Automata.

The intuitive meaning of the concept of n-variable strong automaton is as follows. The inputs for such an automaton are n-tuples of words over Σ. Given an input, the automaton acts on each component of the input exactly as a finite automaton. Computations at different components of the input are independent.

Definition 1.1 *A* **strong n-variable automaton** Ω_n *on* Σ *is a system* (S, S_0, Δ, F)*, where* S*,* S_0*,* Δ *are as for finite automata and* F *is a subset of* S^n*, called the set of* **final states.**

Let $(\alpha_1, \ldots, \alpha_n) \in (\Sigma^\star)^n$ and let α_i be $\sigma_{i1} \ldots \sigma_{im_i}$, $i = 1, \ldots, n$. A sequence $(s_{11}, \ldots, s_{n1}), (s_{12}, \ldots, s_{n2}), \ldots, (s_{1m_1}, \ldots, s_{nm_n})$ is a **computation** of the automaton Ω_n on $(\alpha_1, \ldots, \alpha_n)$ if $(s_{11}, \ldots, s_{n1}) \in S_0^n$, and for each i, the sequence $s_{i1}, \ldots, s_{im_i}$ is a computation of Ω_n on the component α_i according to the transition table. The n-tuple $(\alpha_1, \ldots, \alpha_n)$ is **accepted** by the n-variable strong automataion Ω_n if there exists a computation

$$(s_{11}, \ldots, s_{n1}), (s_{12}, \ldots, s_{n2}), \ldots, (s_{1m_1}, \ldots, s_{nm_n})$$

of the automaton on the n–tuple such that $(s_{1m_1}, \ldots, s_{nm_n}) \in F$. A relation L of airity n on $\Sigma^\star$ is **strongly recognizable** if there exists an n-variable strong automaton Ω_n on Σ such that the set of all n-tuples accepted by this automaton is exactly L.

2. n-variable Automata. Let Σ be a finite alphabet. Suppose that the symbol $\diamond$ does not belong to Σ. Take words $\alpha_i = \sigma_{i1} \ldots, \sigma_{in_i}$ of the alphabet Σ, where $i = 0, \ldots, n-1$. The **convolution** $\alpha_0 \star \ldots \star \alpha_{n-1}$ of these words is defined in the following way. If for all $i, j < n$ $n_i = n_j$, then the convolution is

$$(\sigma_{01}, \ldots, \sigma_{n-11}), \ldots, (\sigma_{0n_0}, \ldots, \sigma_{n-1n_0}).$$

Otherwise, let m be the maximal length of the words $\alpha_0, \ldots, \alpha_{n-1}$. Add to the right end of each α_i the necessary number of symbols $\diamond$ to get words of the length m. Call these new words α_i', $i = 0, \ldots, n-1$. The **convolution** of these n-tuples is $\alpha_0' \star \ldots \star \alpha_{n-1}'$. This convolution is a word of the alphabet $(\Sigma \cup \{\diamond\})^n$. Thus, for any n–ary relation R on $\Sigma^\star$ we can consider the subset $R^\star \subset (\Sigma \cup \{\diamond\})^n$ obtained from R using convolution, that is,

$$R^\star = \{\alpha_0 \star \ldots \star \alpha_{n-1} \mid (\alpha_0, \ldots, \alpha_{n-1}) \in R\}.$$

Definition 1.2 1) **An n-variable automaton** *on* Σ *is a finite automaton over the alphabet* $(\Sigma \cup \{\diamond\})^n$. 2) *An n-ary relation R in $\Sigma^\star$ is* **n-recognizable,** *if $R^\star$ is recognizable by an n–variable automaton.*

3. Asynchronous Automata. Another recognizability notion for relations on $\Sigma^\star$ based on the notion of an asynchronous automaton. Let $P(Q)'$ be the set of all non-empty subsets of Q, let $P(i)'$ be short for $P(\{1, \ldots, i\})'$, $\diamond \notin \Sigma$, and let $\Sigma' = \Sigma \cup \{\diamond\}$.

Definition 1.3 An n-variable asynchronous atomation Ω *on* Σ *is a quadruplet* (S, S_0, Δ, F), *where* S *is* **the set of states**, S_0 *is the set of* **initial states**, $F \subseteq S$ *is* **the set of final states**, *and* $\Delta : S \times (\Sigma')^n \to P(S) \times P(P(n)'))$ *is a partial mapping called the* **transition table** *such that:*

1. *For all* $\sigma \in (\Sigma')^n$, $s \in S$ *if* $\sigma = (\sigma_1, \ldots, \sigma_n)$, $\sigma_i = \diamond$ *for some* i, $\Delta(s, \sigma) = (L, R)$, *and* $J \in R$, *then* $i \notin J$.

2. *For all* $s \in S$, $\Delta(s, \sigma)$ *is undefined if and only if* $\sigma = (\diamond, \diamond, \ldots, \diamond)$.

Let us take words $\alpha_i = \sigma_{i1}, \ldots, \sigma_{in_i}$, $i = 1, \ldots, n$. The intended behaviour of an asynchronous automaton Ω on the n-tuple $\alpha = (\alpha_1, \ldots, \alpha_n)$ is as follows. The automaton begins its computation from an initial state. Suppose that that automaton is in a state $s \in S$ and that the input is $\sigma' = (\sigma_1, \ldots, \sigma_n) \in (\Sigma')^n$. Consider the pair (L, R) defined by the transition table $\Delta(s, \sigma') = (L, R)$. Then the automaton non-deterministically chooses a state $s' \in L$, a non-empty set $\{i_1, \ldots, i_k\} \in R$, and makes moves on components $\alpha_{i_1}, \ldots, \alpha_{i_k}$. We call the pair (s, s') an **elementary move** defined by σ'. Then one can naturally define the notion of a **computation** of the automaton Ω on input α. Say that Ω **accepts** α if there exists a computation of Ω on α which begins at an initial state and ends at a final state. Thus a n-ary relation R on $\Sigma^\star$ is **asynchronously recognizable** if there exists an asynchronous automaton such that the set of all n-tuples of words accepted by this automaton is exactly R.

Presentations of Structures. Let a structure $\mathcal{A}$ of the signature be given. The following are definitions of automatic presentations. The next section gives many examples.

Definition 1.4 *Let* $\nu : D \to A$ *be a surjective mapping, where* D *is an automaton recognizable subset of* $\Sigma^\star$. *The mapping* ν *is respectively an* **automatic, (strong automatic, asynchronous automatic) presentation of** $\mathcal{A}$ *if* ν *satisfies the following conditions:*

1. *There exists a 2–variable automaton (2–variable strong automaton, asynchronous automaton) which for any two words* α, $\beta \in D$ *decides whether* $\nu(\alpha) = \nu(\beta)$.

2. *For each* $j \in \{0, \ldots, s\}$, *there exists an* m_j*–variable automaton (*m_j*–variable strong automaton, asynchronous automaton) which for all* $\alpha_1, \ldots, \alpha_{m_j} \in D$, *decides whether* $P_j^{m_j}(\nu(\alpha_1), \ldots, \nu(\alpha_{m_j}))$ *holds in the structure* $\mathcal{A}$.

3. *For each* $i \in \{0, \ldots, k\}$, *there exists an* $(n_i + 1)$*–variable automaton (*(n_i+1)*–variable strong automaton, asynchronous automaton) which, for all* $\alpha_1, \ldots, \alpha_{n_i}, a_{n_i+1} \in D$, *decides whether* $F_i^{n_i}(\nu(\alpha_1), \ldots, \nu(\alpha_{n_i+1}))$ *holds in* $\mathcal{A}$.

If ν *is an automatic (strongly automatic, asynchronous automatic) presentation of the structure* $\mathcal{A}$, *then the pair* $(\mathcal{A}, \nu)$ *is a* **(strongly, asynchronous) automatic structure** *and the structure is* $\mathcal{A}$
(strongly, asynchronous) automata presentable.

2 Some Examples

Structures with Unary Predicates. These are structures of the form $\mathcal{A} = (A, P_0, \ldots, P_m)$, where each P_i is a unary predicate.

Proposition 2.1 *Every structure with unary predicates only has an automatic presentation.*

Proof. Let $\mathcal{A} = (A, P_0, \ldots, P_m)$ be a structure with each P_i a subset of domain A. Suppose that for different $i, j \leq m$, $P_i \cap P_j = \emptyset$. Consider the alphabet $\Sigma = \{0,1\}$. On the set $\omega = \{0,1\}^\star$ we can choose pairwise disjoint recognizable sets $S_0, \ldots, S_m$ such that for each $i \leq m$, $card(P_i) = card(S_i)$, and

$$card(\{0,1\}^\star \setminus (S_0 \cup \ldots \cup S_m)) = card(A \setminus (P_0 \cup \ldots \cup P_m)).$$

Then any 1-1 function from $\nu : \{0,1\}^\star \to A$ such that $\nu(S_i) = P_i$ for each i, is an automatic presentation of $\mathcal{A}$.

Suppose that $P_0, \ldots, P_n$ are arbitrary unary predicates. Then there exist pairwise disjoint subsets $B_0, \ldots, B_k$ of A with the following property: For any $i \leq m$ there exists a Boolean combination $\Phi_i(B_0, \ldots, B_k)$ of sets $B_0, \ldots, B_k$ such that $P_i = \{x \mid x \in \Phi_i(B_0, \ldots, B_k)\}$. By the previous case, the structure $(A; B_0, \ldots, B_k)$ has a automatic presentation $(\omega; S_0, \ldots, S_k)$. Since recognizable sets are closed under Boolean operations the structure

$$(\omega, \Phi_0(S_0, \ldots, S_k), \ldots, \Phi_m(S_0, \ldots, S_k))$$

is automatic and isomorphic to $\mathcal{A}$. □

Linear Orderings. Here are several examples of automaton presentable linear orderings.

Proposition 2.2 *The rational numbers with the natural linear ordering have an automatic presentation.*

Proof. Let $\Sigma = \{0,1\}$ and let D be a set such that $\alpha 101 \in D$ if and only if $\alpha \in \Sigma^\star$ and α does not have the subword 101. It is clear that D is a recognizable subset of $\Sigma^\star$. Consider the lexicorgraphic linear ordering $\preceq_l$ on the set $\Sigma^\star$. This ordering is recognizable by a 2–variable automaton. Thus the linear ordered set $(D, \preceq_l)$ is automatic. Let $\alpha 101 \in D$. Then $\alpha 101 \preceq_l \alpha 1101$ and $\alpha 00101 \preceq_l \alpha 101$. Hence $(D, \preceq_l)$ does not have maximal and minimal elements. Similarly it can be proved that $\preceq_l$ is a dense linear ordering of the set D. It follows that $(D, \preceq_l)$ is isomorphic to the the rational numbers with the natural linear ordering.□

Proposition 2.3 1) *For any natural number $n \in \omega$, the ordinal ω^n has an automatic presentation.* 2) *The ordinal ω^ω has an asynchronous automatic presentation.*

Proof. 1. Consider the alphabet $\Sigma = \{0,1\}$. To prove the first part, define the following set $D_n = \{0^{i_1}10^{i_2}1 \ldots 0^{i_{n-1}}10^{i_n} \mid i_1, \ldots, i_n \geq 1\}$. The set D_n is recognizable. Let $\alpha, \beta \in \Sigma^\star$. Then $\alpha \leq_n \beta$ if and only if there exist $\gamma, \gamma_1, \gamma_2 \in \Sigma^\star$ such that $\alpha = \gamma 1 \gamma_1$ and $\beta = \gamma 0 \gamma_2$. One can construct a 2–variable automaton which recognizes the relation $\leq_n$. The linearly ordered set $(D, \leq_n)$ is isomorphic to ω^n.

To prove the second part, define the following set D:

$$D = \{0^{i_1}10^{i_2}1 \ldots 0^{i_{k-1}}10^{i_k} \mid i_1, \ldots, i_k \geq 1, k \geq 1\}.$$

Note that D contains the set D_n as a proper subset and that $D = \bigcup_n D_n$. Let $\alpha, \beta \in \Sigma^\star$. Then $\alpha \leq \beta$ if, and only if, either $\alpha \leq_n \beta$ for some n or $\alpha \in D_t$, $\beta \in D_m$, and $t < m$. The linearly ordered set $(D, \leq)$ is isomorphic to ω^ω. It can be verified that the relation $\leq$ is recognizable by a asynchronous automaton. □

The next proposition shows that the standard operations $+$ and $\times$ over linear orderings are preserved by automatic presentations. We leave a proof of this proposition to the reader.

Proposition 2.4 *Let L_1 and L_2 be linear orderings which have (asynchronous) automatic presentations. Then the linear orderings $L_1 + L_2$ and $L_1 \times L_2$ also have (asynchronous) automatic presentations.*□

Boolean Algebras. Let $(L, \leq)$ be a linear ordering. Consider the interval Boolean algebra $\mathcal{B}_L$ generated by **intervals** $[a, b) = \{x \mid a \leq x < b\}$. Any element of this algebra is a finite union of pairwise disjoint intervals. Consider the linear ordering $n \times \omega$, where n is a natural number.

Proposition 2.5 *For every n the Boolean Algebra $\mathcal{B}_{n \times \omega}$ has an asynchronous automatic presentation.*

Proof. We prove the proposition in the case $n = 2$. The remaining case $n > 2$ is similar. We have to prove that the Boolean Algebra $\mathcal{B}_{\omega+\omega}$ has an asynchronous automatic presentation. Consider the alphabet $\{0, 1, 2, \mathbf{0}, \mathbf{1}\}$. Let

$$a = [a_1, b_1) \cup [a_2, b_2) \cup \ldots \cup [a_n, b_n)$$

be an element of the algebra such that $[a_i, b_i) \cap [a_j, b_j) = \emptyset$ for $i \neq j$, and $a_1 \leq b_1 \leq \ldots \leq a_n \leq b_n$. There are several cases.

Case 1. Suppose that $b_n < \omega$. Consider the sequence

$$\nu(a) = \epsilon_0 \ldots \epsilon_{b_n} \in \{0, 1, 2\}^\star,$$

where $\epsilon_i = 1$ if $i \in \bigcup_{j=1}^n [a_j, b_j)$, and $\epsilon_i = 0$ otherwise.

Case 2. Suppose that $a_1 > \omega$. Consider the sequence

$$\nu(a) = 02\epsilon_0 \ldots \epsilon_{b_n} \in \{0, 1, 2\}^\star,$$

where $\epsilon_i = 1$ if $\omega + i \in \bigcup_{j=1}^n [a_j, b_j)$, and $\epsilon_i = 0$ otherwise.

Case 3. Suppose that there exists an $m < n$ such that $b_m < \omega$ and $a_{m+1} > \omega$. Thus

$$a = [a_1, b_1) \cup [a_2, b_2) \cup \ldots \cup [a_m, b_m) \cup \ldots \cup [a_n, b_n).$$

Define $a_1 = [a_1, b_1) \cup \cdots \cup [a_m, b_m)$ and $a_2 = [a_{m+1}, b_{m+1}) \cup \cdots \cup [a_n, b_n)$. Consider the sequence

$$\nu(a) = \nu(a_1)02\nu(a_2).$$

Case 4. Suppose that there exists an $m < n$ such that $a_m < \omega$ and $b_{m+1} > \omega$. Thus $a = [a_1, b_1) \cup [a_2, b_2) \cup \ldots \cup [a_m, b_m) \cup \ldots \cup [a_n, b_n)$. Define

$$\begin{aligned} a_1 &= [a_1, b_1) \cup \ldots \cup [a_m, a_m + 1), \\ a_2 &= [\omega + 1, b_m) \cup [a_{m+1}, b_{m+1}) \cup \ldots \cup [a_n, b_n). \end{aligned}$$

Consider the sequence $\nu(a) = \nu(a_1)12\nu(a_2)$.

We also put $\nu(\emptyset) = \mathbf{0}$ and $\nu(L) = \mathbf{1}$. Thus we have a mapping ν mapping Boolean algebra $\mathcal{B}_{\omega+\omega}$ into $\{0, 1, 2\}^\star$. This mapping is 1-1. Let D be its range. The definition of D implies that the set D is finite automaton recognizable. The operations $\cap$ and $\cup$ in the Boolean Algebra $\mathcal{B}_{\omega+\omega}$ induce the operations $+$ and $\cdot$ in the set D. It is easy to prove that the graphs of $+$ and $\cdot$ on D are recognizable by an asynchronous automaton. □

Graphs. Here we present a general construction of automatic graphs. Suppose that $T = (q_0, Q, P_T)$ is a Turing machine over the finite alphabet $A = \{a_0, \ldots, a_n\}$, where Q is the set of states, P_T is the set of commands of T, q_0 is the initial state. Define the following set D_T:

$$D_T = \{\alpha \mid \alpha \text{ is a configuration of the Turing machine } T\}.$$

The set D_T is a finite automaton recognizable. On the set D_T consider a binary relation R_T defined by

$$R_T = \{(\alpha, \beta) \mid \text{there exists a command in } P_T \text{ transforming } \alpha \text{ to } \beta\}.$$

The set R_T is recognizable by a 2–variable automaton. Define the graph $\mathcal{G}_T = (D_T, R_T)$. We get the following proposition.

Proposition 2.6 *The graph $\mathcal{G}_T = (D_T, R_T)$ is automatic.*□

Unary Structures. We consider structures $(A, f_1, \ldots, f_n)$, where each f_i is a unary operations on A. Here are two results on automatic presentations of unary structures.

Proposition 2.7 *Any free unary structure $\mathcal{A}$ has an automatic presentation.*

Proof. We introduce the set $X = \{0, 00, 000, \ldots, 0^n, \ldots\}$. Let Σ be $\{0, f_1, \ldots, f_n\}$. Define the following set D over this alphabet:

$$D = \{0^n\alpha \mid n \geq 1 \& \alpha \in \{f_1, \ldots, f_n\}^\star\}.$$

The set D is a finite automaton recognizable. Each f_i defines a unary operation, also denoted by f_i, by letting $f_i(0^n\alpha) = 0^n\alpha f_i$. It is clear that $(D, f_1, \ldots, f_n)$ is the free unary algebra with the set of generators X. By the definition of f_i, we conclude that f_i is recognizable by a 2–variable automaton. This proves the proposition.□

A unary structure $(A, f_1, \ldots, f_n)$ is **abelian** if for all $a \in A$, $i, j \leq n$, we have $f_i f_j(a) = f_j f_i(a)$.

Proposition 2.8 *Any free abelian unary structure has an automatic presentation.*

Proof. We introduce the following alphabet $\Sigma = \{0, a_1, \ldots, a_n\}$. Define the set $D = \{0^s a_1^{n_1} \ldots a_n^{i_n} \mid s, i_1, \ldots, i_n \geq 1\}$. The set D is finite automaton recognizable. For each $i \leq n$, define a unary operation f_i on the set D as follows: $f_i(x) = 0^s a_1^{n_1} \ldots a_i^{n_i+1}$ if and only if if $x = 0^s a_1^{n_1} \ldots a_i^{n_i} \ldots a_n^{i_n}$.
It is easy to see that f_i is recognizable by a 2–variable automaton. Therefore the unary structure $(D, f_1, \ldots, f_n)$ is automatic. This structure is the free abelian unary structure on the set $\{0^s \mid s \geq 1\}$ of generators.□

A **permutation structure** is a unary structure $\mathcal{A} = (A, f)$, where f is a 1-1 function defined on the set A.

Proposition 2.9 *If the length of finite cycles of f is bounded, then the permutation structure $\mathcal{A} = (A, f)$ has an automatic presentation.*

Proof. Suppose we are given a permutation structure $\mathcal{A} = (A, f)$ which satisfies the conditions of the proposition. First, suppose that f does not have any cycles of finite length. Consider the alphabet $\{0, 1, 2\}$. Define the following set D which is finite automaton recognizable:

$$D = \{1^n 0^k \mid n, k \geq 1\} \cup \{0^m 2^t \mid m \geq 1, t \geq 0\}.$$

Define on this set unary operation f' as follows.

$$f'(x) = \begin{cases} 1^{n-1}0^k & \text{if } x = 1^n 0^k \text{ and } n \geq 1, \\ 0^m 2^{t+1} & \text{if } x = 0^m 2^t \text{ and } t \geq 0. \end{cases}$$

The function f' is recognizable by a 2–variable automaton. Thus, (D, f') is a permuation structure which has infinitely many infinite cycles. Moreover (D, f') does not have cycles of finite length. From this we conclude that any permutation structure which does not have cycles of finite length has an automatic presentation.

Next, suppose that the length each cycle of $\mathcal{A} = (A, f)$ is n. Consider the set $D \subset \{0, 1\}^\star$ defined by $D = \{0^m 1^i \mid m \geq 1, i = \{1, 2, \ldots, n-1\}\}$. The set D is a finite automaton recognizable. Define f' as follows:

$$f'(x) = \begin{cases} 0^m 1^{i+1} & \text{if } x = 0^m 1^i \text{ and } i < n-1 \\ 0^m & \text{if } x = 0^m 1^{n-1}. \end{cases}$$

The function f' is recognizable by a 2–variable automaton. Thus (D, f') is a permuation structure which is isomorphic to $\mathcal{A}$.

Now consider the general case. Let $(n_1, k_1), \ldots, (n_m, k_m)$ be the sequence of all pairs such that for each $i \leq m$, $n_i, k_i \leq \omega$, and the permutation structure $\mathcal{A} = (A, f)$ has exactly k_i cycles of length n_i. Combining the previous cases, we can conclude that $\mathcal{A}$ has an automatic presentation.□

Is the hypothesis of the previous proposition necessary? Here is an example which shows that this is not a case.

Proposition 2.10 *There exists an automaton presentable permutation structure such that the set of lengths of finite cycles of this structure is not bounded.*

Proof. Define the following function f on set $\{0,1\}^\star$. If $\alpha = 1^n$, then $f(\alpha) = 0^n$. Suppose that $\alpha = \beta 01^n$, where $n \geq 1$. Then $f(\alpha) = \beta 10^n$. Suppose that $\alpha = \beta 0$. Then $f(\alpha) = \beta 1$. One can check that f is recognizable by a 2–variable automaton. Note that, for each n the function forms a cycle of length 2^n.□

Vector Spaces and Abelian Groups. First consider the simplest infinite abelian group, that is, the rank-one free abelian group of rational integers $(Z, +)$.

Lemma 2.1 *The group $(Z, +)$ has an automatic presentation.*

Proof. Each integer $n \in Z$ is a word over the alphabet $\Sigma = \{0, 1, \ldots, 9, -\}$. The standard algorithm which adds two integers gives a 3-variable automaton over Σ recognizing the relation $\{(x, y, z) \mid x + y = z\}$.□

It can be proved that the direct product of any two automata presentable groups is also automaton presentable. Since any finitely generated abelian group can be written as a direct product of a finite group and finitely many copies of $(Z, +)$, we get the following proposition.

Proposition 2.11 *Any finitely generated abelian group has an automatic presentation.* □

Remark. Any finitely generated abelian $\mathcal{A} = (A, +)$ group with the generators $g_1, \ldots, g_n$ induces a unary structure $(A, f_1, \ldots, f_n)$, where $f_i(a) = a + g_i$. In [6] it is proved that $(A, f_1, \ldots, f_n)$ has an automatic presentation. That is, any finitely generated abelian group is automatic in sense of Epstein, Cannon, and Thurston. The proposition above is clearly a stronger version of this result.

Let $\mathcal{V} = (V, \oplus)$ be a vector space over field $\mathcal{F}$. Each $f \in \mathcal{F}$ induces a unary operation, also denoted by f, on the set V by setting $f(v) = vf$. Thus, we can identify the vector space $\mathcal{V}$ with the structure $(V, \oplus, f)_{f \in F}$.

Proposition 2.12 *Any countable vector space $(V, \oplus, f)_{f \in F}$ over a finite field $\mathcal{F}$ has an automatic presentation.*

Proof. Since $\mathcal{F}$ is a finite field, we can suppose that

$$\mathcal{F} = (\{0, 1, \ldots, p-1\}, +, \cdot),$$

where p is a prime number. Consider the set $D = \{0, 1, \dots, p-1\}^\star \setminus \{e\}$, where e is the empty word. Define operation $\oplus$ on this set

$$i_1 \dots i_n \oplus j_1 \dots j_m = \begin{cases} i_1 + j_1 \dots i_n + j_n \dots j_m & \text{if } n \leq m \\ i_1 + j_1 \dots i_m + j_m \dots i_m & \text{if } m \leq n \end{cases}$$

For each $k \in \{0, 1, \dots, p-1\}$ define unary operation f_k by

$$f_k(i_1 \dots i_n) = i_1 \cdot f \dots i_k \cdot f.$$

It can be seen that the relations corresponding to the operations $\oplus$ and f_k are recognizable by 3 and 2–variable automata, respectively.□

3 A Characterization of Automatic Structures

Suppose that $(D, P_0^{m_0}, \dots, P_s^{m_s})$ is a relational structure such that $D \subset \Sigma^\star$, $P_j^{m_j} \subset (\Sigma^\star)^{m_j}$, where $j = 1, \dots, s$, and Σ is a finite alphabet. In this section we present an answer to the following question. When is this structure automatic? In order to answer to this question, we refine the Myhill-Nerode theorem which characterizes finite automaton recognizable languages. It suffices to characterize relations of arity n recognizable by n-variable automata.

Suppose that R is a relation of arity n. Let $R^\star$ be the language over alphabet $(\Sigma \cup \{\diamond\})^n$ obtained from R by convolution. By the definition of automatic structure, structure (D, R) is automatic if and only if D is finite automaton recognizable and the convolution $R^\star$ is recognizable by a finite automaton over alphabet $(\Sigma \cup \{\diamond\})^n$. Thus the Myhill-Nerode theorem can be applied to characterize when structure (D, R) is automatic. But the finite automaton recognizable languages obtained by the convolution of the relations of arity n is a **proper** subclass of all finite automata recognizable languages over $\Sigma \cup \{\diamond\})^n$. The original Myhill-Nerode theorem does not give a characterization of this class. But below, using the idea behind the Myhill-Nerode theorem combined with many-sorted algebra, we give a self-contained characterization of automaton recognizable relations on $\Sigma^\star$. We remark that there have been deep and interesting investigations to characterize this class of relations [19]. Apparently our characterization of this class of such relations is different from previous ones, possibly clearer and simpler.

Let Σ be a finite alphabet. Let $\leq$ be a pre-partial ordering (reflexive and transitive binary relation) on Σ. We say that elements σ_1, σ_2 **have the same sort** if $\sigma_1 \leq \sigma_2$ and $\sigma_2 \leq \sigma_1$. Thus, the elements of Σ are sorted. Moreover, since $\leq$ is a pre-partial ordering, Σ is a finite disjoint union

$$\Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_k,$$

where each Σ_i contains all elements of Σ of the same sort. This induces a partial order on

$$(\{\Sigma_i \mid i = 1, \dots, k\}, \leq).$$

If $\sigma \in \Sigma_i$, then we say that σ has sort i, and we denote it by $s(\sigma)$. If $\sigma_1 \in \Sigma_i$, $\sigma_2 \in \Sigma_j$ and $\Sigma_i \leq \Sigma_j$, we say that the sort i is **weaker** then the sort j. Introduce the following system $\Sigma_{\leq}$:

$$(\Sigma, \leq, \Sigma_1, \ldots, \Sigma_k).$$

Let Σ_1 be the smallest element with respect to $\leq$. We call such a system a **many-sorted finite alphabet.**

Let $\Sigma_{\leq}$ be a many-sorted finite alphabet. Define the set $\Sigma_{\leq}^{\star}$ of **sorted finite words** as follows. The word $\sigma_1 \ldots \sigma_n$ belongs to $\Sigma_{\leq}^{\star}$ if, and only if, for each $i \leq n-1$, the sort of σ_i is weaker than the sort of σ_{i+1}. We take it that the empty word has the weakest sort. Since the alphabet Σ is sorted we can also sort the words in $\Sigma_{\leq}^{\star}$. The **sort of a word** $\alpha \in \Sigma_{\leq}^{\star}$ is the sort of the last symbol appearing in α. Let S_i be the set of all words of sort i. We call any subset of $\Sigma_{\leq}^{\star}$ a **many-sorted language.** We introduce the **many sorted algebra**

$$\mathcal{F} = (S_1, S_2, \ldots, S_k, f_{\sigma})_{\sigma \in \Sigma}),$$

where the unary operation f_{σ} is defined on α and equal to $\alpha\sigma$ if and only if the sort of α is weaker than the sort of σ. We can also define the **many-sorted semigroup**

$$(S_1, S_2, \ldots, S_k, \cdot),$$

where the binary operation $\cdot$ is defined as follows. Let $\alpha, \beta \in \Sigma_{\leq}^{\star}$. Then $\alpha \cdot \beta$ is defined and equal to $\alpha\beta$ if and only if $\alpha\beta \in \Sigma_{\leq}^{\star}$, that is, the sort of α is weaker than the sort of the first letter of β.

Definition 3.5 *A* **many-sorted (non-deterministic) finite automaton** *over the alphabet $\Sigma_{\leq}$ is a system $(Q_1, \ldots, Q_k, I, \Delta, F)$, is defined as follows.*

1. *Each Q_i is the finite* **set of states of sort** *i and $I \subset Q_1$ is the* **set of initial states,**
2. *For all distinct i, j, $Q_i \cap Q_j = \emptyset$.*
3. *$F \subset Q_1 \cup \ldots \cup Q_k$ is the set of* **final states,**
4. *$\Delta \subset (Q_1 \cup \ldots \cup Q_k) \times \Sigma \times (Q_1 \cup \ldots \cup Q_k)$ is the* **transition table** *with the following property. If $(q, \sigma, q') \in \Delta$, then $s(q) \leq s(\sigma) \leq s(q')$.*

If the condition $(q, \sigma, q') \in \Delta$, $(q, \sigma, q'') \in \Delta$ implies that $q' = q''$, then Ω is called **deterministic.**

We can define the notion of computation of many sorted finite automata on sorted words. For a many sorted finite automaton Ω, we can define the set $L(\Omega)$ of all sorted words accepted by the automaton. By the definition of many-sorted automaton, a word accepted by the many-sorted automaton must belong to $\Sigma_{\leq}^{\star}$. We call the set $L(\Omega)$ **recognizable by the many-sorted automaton.** The following lemma can be proved using the standard methods of finite automata theory.

Lemma 3.2 *The set of many-sorted finite automaton recognizable sets is closed under intersection and union.*□

Though the next lemma also uses the known methods of finite automata theory, we present a brief proof of the lemma.

Lemma 3.3 *For any many-sorted finite automaton Ω_1, there exists a deterministic many-sorted finite automaton accepting the same language accepted by Ω_1.*

Proof. Let $\Omega_1 = (Q_1, \ldots, Q_k, I, \Delta, F)$. Define the following many-sorted finite automaton Ω_2:

1. For each i the states of sort i are the subsets of Q_i.
2. The set of initial states contatins only one element which is I.
3. The set of final states consists of all subsets intersecting F.
4. The transition table contains all such triples (Q, σ, Q') such that Q and Q' are the states of the new automaton and the following holds:
   (a) For any $q \in Q$ there exists $q' \in Q'$ for which $(q, \sigma, q') \in \Delta$.
   (b) For any $q' \in Q'$ there exists $q \in Q$ for which $(q, \sigma, q') \in \Delta$.

Then Ω_2 is a many-sorted deterministic finite automaton and accepts exactly those words accepted by the original automaton Ω_1.□

Lemma 3.4 *For any many-sorted finite automaton Ω_1, there is a many-sorted finite automaton accepting the language $\Sigma_{\leq} \setminus L(\Omega_1)$.*

Proof. By the previous lemma we may assume that $\Omega_1 = (Q_1, \ldots, Q_k, I, \Delta, F)$ is deterministic. Thus $\Omega_2 = (Q_1, \ldots, Q_k, I, \Delta, F^c)$ accepts the complement of $L(\Omega_1)$.□

As a corollary of the previous lemmas we get the following theorem.

Theorem 3.1 *The set of all many-sorted finite automata recognizable langauges of $\Sigma_{\leq}$ forms Boolean algebra.*□

Let $\mathcal{F} = (S_1, S_2, \ldots, S_k, f_\sigma)_{\sigma \in \Sigma}$ be the above defined many-sorted algebra. An equivalence realtion $\eta \subset \Sigma_{\leq}^{\star}$ is called a **congruence** if it satisfies the following conditions:

1. For all $(\alpha, \beta) \in \eta$, the words α and β have the same sort.
2. For all $\sigma \in \Sigma$ and $(\alpha, \beta) \in \eta$, if $f_\sigma(\alpha)$ is defined, then $(f_\sigma(\alpha), f_\sigma(\beta)) \in \eta$.

If η is a congruence relation on the many-sorted algebra $\mathcal{F}$, then η is also a **right congruence relation** of the many-sorted semigroup $(S_1, \ldots, S_k, \cdot)$. That is, η satisfies the following condition: for all $(\alpha, \beta) \in \eta$ and $u \in \Sigma_{\leq}^{\star}$, if $\alpha \cdot u$ is defined, then $(\alpha u, \beta u) \in \eta$.

Let L be a many-sorted language over $\Sigma_{\leq}^{\star}$. Define the equivalnce relation η_L as follows. For all α, $\beta \in \Sigma_{\leq}^{\star}$, α and β are η_L–equivalent if

1. α and β have the same sort, and
2. for all $u \in \Sigma_{\leq}^{\star}$ $\alpha \cdot u \in L$ if and only if $\beta \cdot u \in L$.

Then η_L is a congruence relation on the algebra $\mathcal{F}$ (equivalently, is a **right congruence relation** of the many-sorted semigroup $(\Sigma_{\leq}^{\star}, \cdot)$). If the alphabet has only one sort, this equivalence is Myhill-Nerode equivalence. The version of the Myhill-Nerode theorem below gives a characterization of many-sorted automata recognizable languages.

Theorem 3.2 *Let L be a many-sorted language over the alphabet $\Sigma_{\leq}$. The following conditions are equivalent:*

1. *The language L is recognizable by a many-sorted finite automaton.*
2. *The langauge L is a union of some equivalence classes of a right congruence relation η of finite index.*

Outline of Proof. Let Ω be a many-sorted automaton accepting L. We may suppose that Ω is deterministic. Then the equivalence relation η_L has finite index and is a right congruence relation. Suppose that L is the union of some equivalence classes of a right congruence relation η of finite index. Define a finite many-sorted automaton accepting L as follows. The initial state of the automaton will be the empty word. The states of sort i are the η–equivalence classes of words of sort i. A state q is a final state of the automaton if q is a subset of L. A pair (q, σ, q') belongs to the transition table of the automaton if $q \cdot \sigma$ belongs to the η–equivalence class q'. One can verify that this automaton is many-sorted and accepts L.□

We apply this theorem to characterize automatic structures. Suppose that A is a finite alphabet and that $\diamond \notin A$. Introduce the alphabet

$$\Sigma = (A \cup \{\diamond\})^n \setminus \{\diamond\}^n.$$

Define a binary relation $\leq$ on the alphabet Σ as follows:

$$(\sigma_1, \ldots, \sigma_n) \leq (\sigma_1', \ldots, \sigma_n')$$

if, and only if, for all $i \in \{1, \ldots, n\}$, the condition $\sigma_i = \diamond$ implies that $\sigma_i' = \diamond$. It follows that $\leq$ is a pre-partial ordering of the alphabet Σ, giving us a many-sorted finite alphabet $\Sigma_{\leq}$. By the definition of the convolution operation, the convolution $R^\star$ of any relation R of arity n is a many-sorted language over $\Sigma_{\leq}$. The previous result implies:

Theorem 3.3 *Let A be a finite alphabet, and let $(D, P_0^{m_0}, \dots, P_s^{m_s})$ be a structure with $D \subset A$ and $P_i^{m_i} \subset (A^\star)^{m_i}$, where $i0, \dots, s$. For each $i \in \{0, \dots, s\}$ consider the many-sorted alphabet $\Sigma_{\leq, i} = (A \cup \{\diamond\})^{s_i} \setminus \{\diamond\}^{s_i}$. Then the following statements are equivalent:*

1. *The structure $(D, P_0^{m_0}, \dots, P_s^{m_s})$ is automatic.*
2. *The set D is a finite automaton recognizable and the convolution of each set $P_i^{s_i}$ is recognizable by a many-sorted finite automaton.*
3. *The set D is a finite automaton recognizable and the convolution of each set $P_i^{s_i}$ is a union of some equivalence classes of a right congruence relation η of finite index on the many-sorted semigroup $(S_1, \dots, S_k, \cdot)$.*□

We apply the discussion above to characterize automatic structures over one letter alphabet $\Sigma = \{1\}$. We identify the set $\Sigma^\star$ with the set ω. For clarity consider structures with domain ω and binary relations, that is, structures of type

$$(\omega, P_0, \dots, P_s),$$

where each P_i is a binary relation. We introduce the 3-sorted alphabet

$$\Sigma_{\leq} = \{(1,1), (1,\diamond), (\diamond,1)\},$$

where $(1,1) \leq (1,\diamond)$ and $(1,1) \leq (\diamond,1)$. Let

$$\Omega = (Q_1, Q_2, Q_3, q_0, \Delta, F)$$

be a 3–sorted finite deterministic automaton over the alphabet described as follows. The set Q_1 can be thought as a graph which forms a loop. All transitions in Q_1 are labelled by $(1,1)$. Each $s \in Q_1$ forms two disjoint loops $L_s^{(1,\diamond)}$ and $L_s^{(\diamond,1)}$. All transitions in $L_s^{(1,\diamond)}$ are labelled by $(1,\diamond)$ and all transitions in $L_s^{(\diamond,1)}$ are labelled by $(\diamond,1)$. We also can assume that if $s, s' \in Q_1$ are different states, then the loops $L_s^{(1,\diamond)}$, $L_s^{(\diamond,1)}$, $L_{s'}^{(1,\diamond)}$, $L_{s'}^{(\diamond,1)}$ are disjoint.

A set $M \subset \omega$ is **an arithmetic progression** if there exist numbers $n_1 < \dots < n_k \in \omega$ such that

$$M = \{n_1, \dots, n_k\} \cup \{n_k t \mid t \in \omega\}.$$

A subset of ω is **automatic** if it is a finite union of arithmetic progressions. It is easy to see that a set is automatic if and only if it is recognizable by a finite automaton over $\{1\}$.

Corollary 3.1 *Let $(\omega, P_0, \dots, P_s)$ be a structure such that each P_i is a binary relation. This structure is automatic if and only if for each $i \in \{0, \dots, n\}$ there exist automatic sets*

$$A_1, B_1, \dots, A_{i_{k_i}}, B_{i_{k_i}},\ C_1, D_1, \dots, C_{i_{t_i}}, B_{i_{t_i}},$$

such that

$$P_i = \bigcup_{j=1}^{k_i}\{(x, x+n) \mid x \in A_j, n \in B_j\} \cup \bigcup_{s=1}^{t_i}\{(x+n, x) \mid n \in C_s, x \in D_s\}.\Box$$

4 Basic Properties of Automatic Presentations

Features of Decidability. Investigation of automaton recognizable relations over $\Sigma^\star$ suggests investigating the corresponding predicate calculus. Thus, if R_1 and R_2 are automaton recognizable, one can define relations corresponding to the expressions $(R_1 \vee R_2)$, $(R_1 \wedge R_2)$, and $\neg(R_1)$, $\exists x R_1$, and $\forall x(R_1)$. For instance, suppose R_1 is an n-ary relation. Define

$$\begin{aligned}
\exists x_i(R_1) &= \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \mid \\
&\qquad (x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) \in R_1\}, \\
\forall x_i(R_1) &= \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \mid \\
&\qquad \forall \alpha\, R(x_1, \ldots, x_{i-1}, \alpha, x_{i+1}, \ldots, x_n)\}.
\end{aligned}$$

When R_1 is a unary relation, then $\exists x(R_1)$ corresponds to $\emptyset$ if $R_1 = \emptyset$, and to $\Sigma^\star$ otherwise. Similarly, $\forall x(R_1)$ corresponds to $\emptyset$ if $R_1 \neq \Sigma^\star$, and to $\Sigma^\star$ otherwise.

The next theorem is a consequence of standard finite automata theory of the middle 1950's..

Theorem 4.4 *1. Let R_1 and R_2 be automata recognizable relations. Then the relations corresponding to the expressions $(R_1 \vee R_2)$, $(R_1 \wedge R_2)$, $\neg(R_1)$, $\exists x(R_1)$, and $\forall x(R_1)$ are also automata recognizable.*

2. The emptiness problem for n-variable automata is decidable uniformly in n.

3. There exists a procedure which, for automata recognizing R_1 and R_2, constructs automata for recognizing the relations corresponding to the expressions $(R_1 \vee R_2)$, $(R_1 \wedge R_2)$, $\neg(R_1)$, $\exists x(R_1)$, and $\forall x(R_1)$. □

This theorem implies several properties of automatic presentations.

Corollary 4.2 *Suppose that a structure $\mathcal{A}$ has an automatic presentation. Then*

1. *There exists an effective procedure which, applied to a first order definition of a relation P on $\mathcal{A}$, yields an algorithm deciding P.*
2. *The first order theory of the structure $\mathcal{A}$ is decidable.*

Proof. Let $\nu : D \to A$ be an automatic presentation. Then the atomic relations and the equality relation on this structure are decidable under the presentation ν. By the theorem just stated, there exists an effective procedure which, for any first order definable relation P, produces an algorithm deciding P. It also follows that the first order theory of this structure is decidable.□

Remark. In fact, if $\mathcal{A}$ is automatic, the corollary above can be strengthened. Namely, consider the set of all polynomials over the structure $\mathcal{A}$. (This is the set of all functions of the form
$t(a_1, \ldots, a_k, x, a_{k+1}, \ldots, a_m)$, where x is a variable, T is a term, and $a_1, \ldots, a_m$ are elements of $\mathcal{A}$.) Using the decidability of the monadic second order theory of two successor functions [12], it follows that the first order theory of $\mathcal{A}$ plus the monadic second order theory of polynomials over $\mathcal{A}$ is decidable [20].

Corollary 4.3 *If structure*

$$\mathcal{A} = (A, f_0^{n_0}, \ldots, f_k^{n_k}, P_0^{m_0}, \ldots, P_s^{m_s}, c_0, \ldots, c_t),$$

has an automatic presentation, then there exists an automatic presentation $\mu : d \to a$ which is a 1-1 function.

Proof. Let $\nu : D_1 \to A$ be an automatic presentation of $\mathcal{A}$. Let $D \subset \{0, 1, \ldots, n-1\}^\star$. Define ordering $\preceq$ on set $\Sigma^\star$ as follows. If the length of α is less than the length of β, then $\alpha \preceq \beta$. If α and β have the same length, then $\alpha \preceq \beta$ if and only if there exist $\gamma, \gamma_1, \gamma_2 \in \Sigma^\star$ such that $\alpha = \gamma i \gamma_1$, $\beta = \gamma j \gamma_2$ and $i < j \leq n-1$. This relation is recognizable by a 2–variable automaton. Moreover $\preceq$ is a linear ordering of $\Sigma^\star$ isomorphic to the natural ordering of ω. Consider the following relation

$$D = \{\alpha \mid \alpha \in D \& \forall \beta (\nu(\alpha) = \nu(\beta) \to \alpha \preceq \beta\}.$$

By the theorem above, the set R is a finite automaton recognizable. For every $i \leq k$ there exists a n_i–variable automaton which recognize the set

$$\{(\alpha_1, \ldots, \alpha_{n_i}, \alpha_{n_i+1}) \mid \mathcal{A} \models F_i^{n_i+1}(\nu(\alpha_1), \ldots, \nu(\alpha_{n_i}), \nu(\alpha_{n_i+1}))\}.$$

By the theorem above, the set

$$\begin{aligned} L_i^{n_i+1} \quad = \quad & \{(\alpha_1, \ldots, \alpha_{n_i+1}) \in D^{n_i+1} \mid \\ & \qquad \mathcal{A} \models F_i^{n_i+1}(\nu(\alpha_1), \ldots, \nu(\alpha_{n_i}), \nu(\alpha_{n_i+1}))\} \end{aligned}$$

is also recognizable by an $(n_i{+}1)$–variable automaton. Similarly, for each $j \leq s$, the set

$$M_i^{m_j} = \{(\alpha_1, \ldots, \alpha_{m_j}) \in D^{m_j} \mid \mathcal{A} \models P_j^{m_j}(\nu(\alpha_1), \ldots, \nu(\alpha_{n_i}))\}$$

is recognizable by an m_j–variable automaton. It follows that the structure

$$(D, L_0^{n_0+1}, \ldots, L_k^{n_k+1}, M_0^{m_0}, \ldots, M_s^{m_s}, c_0, \ldots, c_t)$$

is isomorphic to $\mathcal{A}$. The mapping $\mu : D \to A$ defined by $\mu(\alpha) = \nu(\alpha)$ is 1-1. □

Using the theorem above, similar to the previous corollary, one can prove that automata presentable structures are closed under direct product and factorizations with respect to 2–variable automata recognizable congruences.

Corollary 4.4 *1. Let $\nu_1 : D_1 \to A_1$ and $\nu_2 : D_2 \to A_2$ be automatic presentations of structures $\mathcal{A}_1$ and $\mathcal{A}_2$ of the same signature. Then the structure $\mathcal{A}_1 \times \mathcal{A}_2$ possesses an automatic presentation.*

2. Let $\nu_1 : D \to A$ be an automatic presentation of $\mathcal{A}_1$. If η is a congruence of $\mathcal{A}$ recognizable by a 2–variable automaton, then the factor structure $\mathcal{A}/\eta$ possesses an automatic presentation. □

Finitely Generated Automatic Structures. Suppose that a structure $\mathcal{A}$ is finitely generated. Let $a_1, \ldots, a_k$ be generators of this structure. Let $f_1^{m_1}, \ldots, f_t^{m_t}$ be all atomic operations of the algebra. We define the following sequence of finite sets generating the structure.

Stage 0. Put $G_0 = \{a_0, \ldots, a_k\}$.
Stage n+1. Suppose that G_t has been defined. Then

$$G_{n+1} = G_n \cup \{f_i^{m_i}(b_1, \ldots, b_{m_i}) \mid i = 1, \ldots, t, b_1, \ldots, b_{m_i} \in G_n\}.$$

Definition 4.6 *Consider the function f defined by $f(n) = card(G_n)$. We call this function the* **grouth level** *of the generators $a_1, \ldots, a_k$.*

Lemma 4.5 *Let $a_1, \ldots, a_k$ be generators of the automatic structure $\mathcal{A}$ over the alphabet Σ of cardinality s. Then there exist $a, b \in \omega$ such that the growth level of the generators does not exceed s^{a+1+bn}.*

Proof. Since $\mathcal{A}$ is automatic, there exist finite automata $\Omega_1, \ldots, \Omega_t$ recognizing the graphs of the operations. Let a be the maximum of the lengths of the generators $a_1, \ldots, a_k$. Let b be an upper bound for the number of states of all automata Ω_i, $i = 1, \ldots, n$. We can prove by induction on n that $card(G_n) \leq s^{a+1+bn}$.

For the base step, if $n = 0$, $card(G_0) \leq s^{a+1}$, since the number of elements of length n does not exceed s^{n+1}.

For the induction step, sppose the conclusion is true for $card(G_n) \leq s^{a+1+bn}$. Let $a = f(b_1, \ldots, b_m)$ for some atomic operation f and $b_1, \ldots, b_m \in G_n$. By inductive hypothesis, the lengths of the b_i do not exceed $a + bn$. It follows that the length of a does not exceed $a + b(n+1)$. Thus G_{n+1} is a subset of the set of all words of length not exceeding $\leq a + b(n+1)$. The cardinality of this set is at most $s^{a+1+b(n+1)}$, as required.□

Corollary 4.5 *Let $\mathcal{A}$ be an automatic structure. For any substructure generated by some elements $a_1, \ldots, a_k$, there exist $s, a, b \in \omega$ such that the growth level of the generators does not exceed s^{a+bn}.*□

Using this lemma one can get many examples of structures which do not have automatic presentations but still possess the features of decidability mentioned above.

Corollary 4.6 *Suppose that f is a function symbol of arity 2 and c is a constant symbol. Then the absolutely free algebra generated by c and f does not have an automatic presentation.*

Proof. The elements of the algebra are the terms defined by the following inductive definition.

1. c is a term.
2. If t_1 and t_2 are terms, then $f(t_1, t_2)$ is a term.

Let T be the set of terms. The free algebra is (T, f). The generator of this algebra is c. The lower bound for the growth level of the generator c is $2^{2^n} - n$. Therefore the absolutely free algebra (T, f) does not have an automatic presentation. □

5 Strongly Automatic Presentations

In this section we give a characterization of strongly automata presentable structures. Let Σ be a finite alphabet, $\diamond \notin \Sigma$ and $n > 1$. Put $\Sigma_\diamond = \Sigma \cup \{\diamond\}$. On the set of $\Sigma_\diamond^\star$ we define a relation $\sim_\diamond$. Let $\alpha, \beta \in \Sigma_\diamond^\star$. Let $\alpha_\diamond$ be the word obtained from α ommiting all occurences of $\diamond$. Define

$$\alpha \sim_\diamond \beta \leftrightarrow \alpha_\diamond = \beta_\diamond .$$

Thus, $\sim_\diamond$ is an equivalence relation on $\Sigma_\diamond^\star$. We can then define an equivalence relation $\sim_\diamond^n$ on the set of all n-tuples of $\Sigma_\diamond^\star$ as follows. Two n-tuples are equivalent if and only if their corresponding components are $\sim_\diamond$-equivalent. By the definition of $\sim_n^\diamond$, that each $\sim_\diamond^n$-equivalence class is represented by some unique n-tuple from the set $(\Sigma^\star)^n$. Thus, there is a natural 1-1 correspondence between the factor set $(\Sigma_\diamond^\star)^n / \sim_\diamond^n$ and $(\Sigma^\star)^n$. Define a binary operation on $(\Sigma^\star)^n$ as follows.

$$(\alpha_1, \ldots, \alpha_n)(\beta_1, \ldots, \beta_n) = (\alpha_1\beta_1, \ldots, \alpha_n\beta_n)$$

This operation is well-defined on $\sim_\diamond^n$-equivalence classes. Let R be a relation on $\Sigma^\star$ of arity n. Define a Myhill-Nerode equivalence relation $\sim_R$ on $(\Sigma^\star)^n$ by:

$$\alpha \sim_R \beta \leftrightarrow \forall u \in (\Sigma^\star)^n (\alpha u \in R \leftrightarrow \beta u \in R)$$

Then $\sim_R$ is an equivalence relation compatible with the right multiplication, that is, if $\alpha \sim_R \beta$, then for all u, $\alpha u \sim_R \beta u$.

Definition 5.7 *An n–variable automaton $\Omega = (S, S_0, \Delta, F)$ on Σ is* **simple** *if:*

1. *For all $s \in S$ and $\alpha = (\alpha_1, \ldots, \alpha_n), \beta = (\beta_1, \ldots, \beta_n) \in (\Sigma_\diamond^\star)^n$, if $\alpha \sim_\diamond^n \beta$, then $\Delta(s, \alpha_1 \star \ldots \star \alpha_n) = \Delta(s, \beta_1 \star \ldots \star \beta_n)$.*
2. *For all $s \in S$, $\sigma = (\sigma_1, \ldots, \sigma_n), \delta = (\delta_1, \ldots, \delta_n) \in (\Sigma_\diamond)^n$, if $\Delta(s, \sigma) = \Delta(s, \delta)$, then $\Delta(s, \sigma) = \Delta(s, (\gamma_1, \ldots, \gamma_n))$, where $\gamma_i \in \{\sigma_i, \delta_i\}$.*

The next theorem characterizes strongly recognizable relations in terms of simple automata, the equivalence $\sim_R$ and finite automata recognizable subsets of $\Sigma^\star$. The **index** of an equivalence relation is the number of equivalence classes.

Theorem 5.5 *Let Σ be an alphabet. Let R be a relation on $\Sigma^\star$ of arity n. The following statments are equivalent:*

1. *R is a strongly n-recognizable.*

2. *R is a union of some classes of an equivalence relation with a finite index and compatible with the right multiplication on $(\Sigma^\star)^n$.*

3. *R is accepted by a simple n-variable automaton over Σ.*

4. *There exists $k \in \omega$ and finite automata recognizable subsets $R_{1i}, \ldots, R_{ni}$ of $\Sigma^\star$, where $1 \leq i \leq k$, such that $R = \bigcup_{i=1}^{k} R_{1i} \times \ldots \times R_{ni}$.*

Proof. 1) → 2). Let $\Omega = (S, S_0, \Delta, F)$ be an n–variable strong automaton accepting R. Define a congruence relation $\sim$ on $(\Sigma^\star)^n$ as follows:

$$(\alpha_1, \ldots, \alpha_n) \sim (\beta_1, \ldots, \beta_n)$$

if, and only if, for all $s, q \in S^n$ and for all $i \leq n$, there exists a computation of Ω on α_i which begins in state q and ends in state s if and only if there exists a computation of Ω on β_i which begins in q and ends in s. The relation $\sim$ is a congruence of a finite index. Since R is recognizable by Ω, by the definition of $\sim$, R is a union of some $\sim$–equivalence classes.

2) → 3). Let $\sim$ be a given equivalence compatible with the right multiplication. Suppose that R is a union of some classes of this equivalence relation. Define an n-variable automaton Ω. Let the set of states to be $S = \{\alpha_\sim^\diamond \mid \alpha \in (\Sigma^\star)^n\}$, where $\alpha_\sim^\diamond$ is the $\sim_\diamond^n$–closure of the $\sim$–equivalence class containing $\alpha \in (\Sigma^\star)^n$; let the set of initial states to be $S_0 = \{(\lambda, \ldots, \lambda)_\sim^\diamond\}$, where λ is the empty word; put $F = \{\alpha_\sim \mid \alpha \in R\}$; define the transition table Δ as follows: for all $\alpha_\sim^\diamond$ and $\sigma \in (\Sigma_\diamond)^n$ put $\Delta(\alpha_\sim^\diamond, \sigma) = (\alpha\sigma)_\sim^\diamond$. This automaton is simple.

3) → 4). Suppose that R is accepted by a simple n–variable automaton $\mathcal{A}$ over Σ. Let $f_1, \ldots, f_k$ be all final states of the automaton $\mathcal{A}$. Let $R(f_i)$ be the set of all n-tuples which transform the initial state q_0 of $\mathcal{A}$ to f_i. Let $R(f_i, 1), \ldots, R(f_i, n)$ be the projections of $R(f_i)$ onto corresponding components. Using the definition of simple automaton, we get $R(f_i) = R(f_i, 1) \times \ldots \times R(f_i, n)$. Thus,

$$R = \bigcup_{i=1}^{k} R(f_i, 1) \times \ldots \times R(f_i, n).$$

Note that for all i, j the set $R(f_i, j)$ is a finite automaton recognizable subset of $\Sigma^\star$.

4) → 1). For any $i, 1 \leq i \leq k$, there exists a finite automaton

$$\Omega_i = (S_i, q_{i0}, \Delta_i, F_{i1} \cup \ldots \cup F_{in})$$

such that for any $\alpha \in \Sigma^\star$, $\alpha \in R_{ij}$ if and only if there exists a computation of Ω_i which begins in q_{i0} and ends in F_{ij}. We can suppose that for all $i \neq j$, $S_i \bigcap S_j = \emptyset$. Define an automaton $\Omega = (S, S_0, \Delta, F)$ as follows.

1. $S = S_1 \cup \ldots \cup S_n$ and $S_0 = \{q_{10}; \ldots, q_{k0}\}$.
2. $F = \bigcup_{i=0}^{k} F_{i1} \times \ldots \times F_{in}$.
3. $\Delta = \bigcup \Delta_i$.

This automaton is an n–variable strong automaton which accepts R. □

This theorem and the theorem of the previous section allow us to obtain a corollary characterizing structures which possess strongly automatic presentations. We need the following notion.

An n–ary relation P on a set A is called **complete** if there exist subsets $A_{i1}, \ldots, A_{in}$ of A and a number $k \in \omega$ such that $P = \bigcup_{i=1}^{k} A_{i1} \times \ldots \times A_{in}$.

Theorem 5.6 *A structure*

$$\mathcal{A} = (A; P_0^{m_0}, \ldots, P_t^{m_t})$$

has a strongly automatic presentation if and only if each predicate $P_i^{n_i}$ is complete.

Proof. For simplicity suppose that $t = 0$ and $m_0 = n$. If $\mathcal{A}$ has a strongly automatic presentation (L, R) then, by the previous theorem, there are recognizable sets $R_{1i}, \ldots, R_{ni}$ such that $R = \bigcup_i R_{1i} \times \ldots \times R_{ni}$.. Hence $P_0^{m_0}$ is a complete relation.

Conversely, suppose that P is a complete relation on A. Then there exist subsets $A_{i1}, \ldots, A_{in}$ and a number $k \in \omega$ such that $P = \bigcup_i A_{i1} \times \ldots \times A_{in}$. Consider a structure $\mathcal{A}_1 = (A; A_{11}, \ldots, A_{1n}, \ldots, A_{k1}, \ldots, A_{kn})$. Structure $\mathcal{A}$ is isomorphic to

$$(A; \bigcup_i A_{i1} \times \ldots \times A_{in}).$$

By Proposition 2.1, $\mathcal{A}_1$ has a strong automatic presentation

$$(L; R_{11}, \ldots, R_{1n}, \ldots, R_{k1}, \ldots, R_{kn}).$$

Let $R = \bigcup_i R_{i1} \times \ldots \times R_{in}$. By the previous theorem, relation R is accepted by an n-variable strong automaton. Thus the structure $(L; R)$ is isomorphic to $\mathcal{A}$. Hence $\mathcal{A}$ possesses a strongly automatic presentation.□

Corollary 5.7 *The structures $(\omega; \leq)$ and $(\omega; s)$ do not possess strongly automatic presentations.*□

6 Automatic Isomorphism Types

A basic problem for recursive and polynomial time structures is to characterize structures which have the same isomorphism type via recursive or p–time computable isomorphisms. A structure is recursively (polynomial time) categorical if any two recursive (p–time) presentations of the structure are recursively (p–time) isomorphic. Thus in some sense a recursively (p–time) categorical structure is one for which the problem of recursive (p–time) presentations has a unique solution. It looks very hard to find general necessary and sufficient conditions for structures to be recursively (p–time) categorical. The corresponding problem for automatic isomorphism types is easy.

Definition 6.8 *We say that sets $R_1, R_2 \subset \Sigma^\star$ have the same* **automatic isomorphism type** *if there exists a relation f, recognizable by a 2–variable automaton, such that $dom(f) = R_1$, $range(f) = R_2$, and f is a 1-1 function. In this case we say that R_1 and R_2 are* **automatically isomorphic** *via the* **automatic isomorphism** *f.*

For an R we denote by $AI(R)$ the class of all sets automatically isomorphic to R. Note that if $L \in AI(R)$, then L is a finite automaton recognizable. It is obvious that if R_1 is a finite set, then $R_2 \in AI(R_1)$ if and only if $card(R_1) = card(R_2)$. Our next result shows that for any infinite recognizable set there exists in some sense a standard presentation of this set which has the same AI type. We need definitions.

Definition 6.9 *A set $A \subset \Sigma^\star$ is a* **free monoid** *if*

1. *There exists a $B \subset A$ such that $B^\star = A$, and*
2. *for all $b_1, \ldots, b_s, a_1, \ldots, a_k \in B$ if $b_1 \ldots b_s = a_1 \ldots a_n$, then $k = n$ and $a_i = b_i$ for all $i = 1, \ldots, n$.*

Let $A, B \subset \Sigma^\star$. The mulitiplication $A \cdot B$ is **free** if for for all $a_1, a_2 \in A, b_1, b_2 \in B$ if $a_1 \cdot b_1 = a_2 \cdot b_2$, then $a_1 = a_2$ and $b_1 = b_2$.

Using Eilenberg's decomposition theorem for finite automata recognizable sets [5], one can characterize automatic isomorphism types for structures in the language of pure equality.

Theorem 6.7 *Let L be a finite automaton recognizable infinite set. There exist finite automata recognizable pairwise disjoint sets $L_{11}, \ldots, L_{1k_1}, \ldots, L_{n1}, \ldots, L_{nk_n}$ such that:*

1. *For all i, j, $i = 1, \ldots, n$, $j = 1, \ldots, k_i$, the set L_{ij} is a free submonoid.*
2. *For all $i \neq j$, $i \leq n$, the multiplication $L_{i1} \cdot \ldots \cdot L_{ik_i}$ is free.*
3. *$L = (L_{11} \cdot \ldots \cdot L_{1k_1}) \cup \ldots \cup (L_{n1} \cdot \ldots \cdot L_{kn_k}) \in AI(R)$, where $AI(R)$ is the automatic isomorphism type of R.*

That is, every finite automata recognizable set is automatically isomorphic to a finite union of free mulitiplications of free monoids.□

Now we investigate the more general problem of automatic isomorphism types of automatic structures.

Definition 6.10 *Let $\mathcal{A}$ be an automatic structure. An automatic structure $\mathcal{B}$ is* **automatically isomorphic** *to $\mathcal{A}$ if there exists a function f, recognizable by a 2–variable automaton, such that dom$(f) = A$, range$(f) = B$ and f induces an isomorphism between these structures.*

Let $\mathcal{A}$ be automaton presentable structure. The number of automatic isomorphism types of $\mathcal{A}$ we call the **automatic dimension** of $\mathcal{A}$. The structure $\mathcal{A}$ is **automatically categorical**if its automatic dimension is 1.

Theorem 6.8 *The automatic dimension of any automaton presentable structure is either ω or 1. Moreover, such a structure is automatically categorical if and only if its domain is finite.*

Proof. Let B be an automaton recognizable set. Let us consider the sequence $b_0 \preceq b_1 \preceq b_2 \preceq \ldots$ of all elements of the set B, where $\preceq$ is the linear ordering on $\Sigma^\star$ defined in the proof of Corollary 4.2. We define a function $f_B(n) = length(b_n)$.

Lemma 6.6 *Let B, C be automaton recognizable sets. If the sequence $|f_B(n) - f_C(n)|$ is increasing, then B and C do not have the same automatic isomorphism type.*

Proof. Suppose that the lemma is not true and B, C be automaton recognizable sets such that the set $\{|f_B(n) - f_C(n)| \mid n \in \omega\}$ is not bounded. Let g be 1-1 function such that $g(B) = C$. Suppose that g is recognizable by a 2-variable automaton. Let n be such that for any $b \in B$, we have $|length(b) - length(g(b))| \leq n$. We may assume that n is the number of states of the automaton recognizing g. Indeed, otherwise, applying the pumping lemma of finite automata theory, we would contradict the fact that g is 1-1. Since the sequence $|f_B(m) - f_C(m)|$ is not bounded, there is an $s \in \omega$ such that $|f_B(s) - f_C(s)| > n$. There are two cases.

Case 1. Suppose that $f_C(s) - f_B(s) > n$. Then $g(b_i) \notin \{c_s, c_{s+1}, \ldots\}$ for all $i \leq s$. Since g is 1-1, we should have $\{g(b_0), \ldots, g(b_{s-1})\} = \{c_0, \ldots, c_{s-1}\}$. But we also have $g(b_m) \in \{c_0, \ldots, c_{s-1}\}$, a contradiction.

Case 2. Suppose that $f_B(s) - f_C(s) > n$. Then $f(b_m) \notin \{c_0, \ldots, c_s\}$ for all $m \geq s$. Thus g is not 1-1, a contradiction. We proved the lemma.

Let $\mathcal{A}$ be an infinite automatic structure. Let c be a new symbol which does not belong to Σ. From the structure $\mathcal{A} = (A; P_0^{k_0}, \ldots, P_t^{k_t})$ we define a new structure $\mathcal{B}_n$:

1. The universe of $\mathcal{B}_n$ is
$$B_n = \{a_0 c^n a_1 c^n \ldots a_m c^n \mid a_0 \ldots a_m \in A\}$$
(Notice if $n = 0$ then $B_n = A$.)
2. For each predicate $P_i^{k_i}$ we define a predicate $Q_i^{k_i}$ as follows. A tuple
$$(a_{01} c^n a_{11} c^n \ldots a_{m_0 1} c^n, \ldots, a_{0k_i} c^n a_{1k_i} c^n \ldots a_{m_i k_i} c^n)$$

belongs to $Q_i^{k_i}$ if and only if $(a_{01} \dots a_{m_0 1}, \dots, a_{1k_i} \dots a_{m_i k_i})$ belongs to $P_i^{k_i}$.

This defines a structure

$$\mathcal{B}_n = (B_n; Q_0^{k_0}, \dots, Q_t^{k_t}).$$

¿From the construction, $\mathcal{B}_n$ is isomorphic to $\mathcal{A}$. Since $\mathcal{A}$ is an automatic structure, it follows that $\mathcal{B}_n$ is also an automatic structure.

The the sequence $|f_{\mathcal{B}_{n-1}}(m) - f_{\mathcal{B}_n}(m)|$ is increasing for any fixed $n \in \omega$. The lemma above implies that the structure B_n is not automatically isomorphic to the structure $\mathcal{B}_{n-1}$. To complete the proof, note that finite structures are automatically categorical. □

7 Conclusion and Open Questions

The theory of automatic structures can be considered as a branch of the theory of recursive structures. But one has to take into account the differences between these two approaches for investigating the connections between algebraic, model-theoretic, and effective properties of structures. Recursive model theory can be viewed as an application of recursion theory to model theory, while the theory of automatic structures can be viewed as an application of complexity theory to model theory.

For example, suppose that we have a structure $\mathcal{A}$ and a relation R which is of particular interest. If we are interesting in deciding R, from the recursive structures point of view, we would consider the following type of questions.

1. Does there exist a recursive copy of $\mathcal{A}$ on which R is decidable?

2. Does there exists a recursive copy of $\mathcal{A}$ on which R is creative set?

3. Does there exist a recursive copy of $\mathcal{A}$ on which R is a simple set, or has a particular Turing or m-degree?

But from the point of view of automatic structures, we would naturally consider the following questions.

1. Does there exist an automatic copy of $\mathcal{A}$?

2. Does there exist a automatic presentation of $\mathcal{A}$ on which R is decidable?

3. Does there exist an automatic presentation of $\mathcal{A}$ in which there is a decision procedure for R of given time or space complexity?

4. Does there exist an automatic presentation of $\mathcal{A}$ in which the decision procedure for R is NP–complete?

This paper suggests that recursive, algebraic, model theoretic, and complexity theoretic properties of automatic structures are amendable to systematic investigation. Here are a few open questions.

Question 1. Characterize the first countable ordinal which does not have any automatic presentation.

In section 2, we gave automatic presentations for ordinals ω^n, $n \in \omega$.

Question 2. Give an algebraic or model-theoretic characterization of automata presentable structures.

In section 5 we characterized strongly automata representable structures in terms of complete relations.

Question 3. Characterize decidable first order theories for which every countable model has an automatic reprsnetation.

Section 4 shows that the first order theory of every automatic structure is decidable. On the other hand it is known that every decidable first order theory possesses a decidable structure [7]. It can also be proved that finding the truth value of a fully quantified automaton recognizable predicate is exponential in the size of the automaton recognizing the relation [6].

Question 4. Characterize automatic isomorphism types of automatic structures over a fixed domain.

In the last section, we proved that if we do not fix domains, then any structure with exactly one isomorphism type is finite. We do not know, however, what effect a fixed domain has on automatic isomorphism types.

In this paper we dis not consider presentations of structures using tree automata. One can develop and study tree automata presentatable structures. Of course, an approach based on tree automata would cover this paper and possess the same positive features of decidability. However we decided to present clear definitions and examples based on the simple computational structure of finite automata. The investigation of tree automata presentable structures we have under development. [20].

References

[1] Aspects of Effective Algebra, Proceedings of a Conference at Monash University, 1979, edited by J.N. Crossley.

[2] C.J. Ash, A. Nerode, Intrinsically Recursive Relations, Aspects of Effective Algebra, Proceeding of a Conference at Monash University, Australia, 1979.

[3] R. Buchi, The Monadic Theory of ω_1, in: Decidable Theories II, Lecture Notes in Mathematics, 328, 1-127.

[4] R. Buchi, D. Siefkes, Axiomatization of the Monadic second Order Theory of ω_1, in: Decidable Theories II, Lect. Not. in Mathematics, 328, 127-215.

[5] S. Eilenberg, Automata, Languages, and Mashines, vol. A, Academic Press, New York, 1974.

[6] D.Epstein, J.Cannon, and others, Word Processing in Groups, Jones and Bartlett Publishers, Boston, London.

[7] Yu.L. Ershov, Problems of Decidability and Constructive Models, Moscow, 1989.

[8] A.Frohlich, J. Shepherdson, Effective Procedures in Field Theory, Philos. Tranns. Roy. Soc., London, ser A 248, 1955, 432-487.

[9] S.S. Goncharov, The Problem of the Number Of Non-Self-Equivalent Constructivizations, Algebra and Logic, No 6, 1980.

[10] Y. Gurevich and L. Harrington, Trees, Automata and Games, Proceedings of the 14th Annual ACM Symposium on Theory of Computing, 1982, 60-65.

[11] A.I. Mal'cev A.I, Constructive Algebras, Uspekhi Matem. Nauk, 16, No 3, 1961.

[12] M. Rabin, Decidability of Second-Order Theories and Automata on Infinite Trees, Trans. of American Math. Soc., 141, 1969, 1-35.

[13] M.Rabin, Weakly Definable Relations and Special Automata, Mathematical Logic and Foundations of Set Theory, North-Holland, Amsterdam, 1970, 1-23.

[14] M. Rabin, Computable Algebra: General theory and Theory of Computable Fields, Trans.Amer. Math. Soc., 95, 1960, 341-360

[15] J.B. Remmel, D. Cenzer, Polynomial Time Versus Recursive Models, Annals of Pure and Applied Logic, 1991.

[16] J.B. Remmel, D. Cenzer, Feasibly Categorical Abelian Groups, Proceedings of the Workshop Feasible Mathematics II, to appear.

[17] H. Rogers, Theory of Recursive Function and Effective Computability, New York, 1967.

[18] W. Thomas, Automata on Infinite Objects, in Handbook of Theoretical Computer Science, Volume B, J. van Leeuwen editor, The MIT Press/Elsevier, 1990, 133-192.

[19] C. Frougny, J. Sakarovitch, Synchronized Ratinal Relations of Finite and Infinite Words, Theoretical Computer Science, 108, 1993, 45-82.

[20] B.Khoussainov, A. Nerode, Automatic Models and $S2S$, in preparation.

A Restricted Second Order Logic for Finite Structures

Anuj Dawar*

Department of Computer Science, University of Wales, Swansea, Singleton Park,
Swansea SA2 8PP, U.K.

Abstract. We introduce a restricted version of second order logic SO^{ω} in which the second order quantifiers range over relations that are closed under the equivalence relation $\equiv^k$ of k variable equivalence, for some k. This restricted second order logic is an effective fragment of the infinitary logic $L^{\omega}_{\infty\omega}$, which differs from other such fragments in that it is not based on a fixpoint logic. We explore the relationship of SO^{ω} with fixpoint logics, showing that its inclusion relations with these logics are equivalent to problems in complexity theory. We also look at the expressibility of NP-complete problems in this logic.

1 Introduction

In recent years, much research in finite model theory has focused on its connections with computational complexity theory. It turns out that there is a close relationship between the computational complexity of a problem, i.e. the amount of resources needed to solve the problem on some machine model of computation, and its descriptive complexity, i.e. the kinds of "logical resources" that are needed to describe the problem. The paradigmatic result establishing a connection between descriptive and computational complexity is the result of Fagin [11] which shows that the properties of finite structures that are definable by sentences of existential second order logic are exactly those that are in the complexity class NP. This was extended by Stockmeyer [23] to a tight correspondence betwen second order logic and the polynomial time hierarchy. Further work along these lines has established logical characterizations for a wide range of complexity classes (see, for instance, [16]).

However, some of the results equating logical expressibility to computational complexity require the finite structures to have a built-in linear order. That is, the exact correspondence between expressibility in a logic and solvability within given resource bounds does not hold over the class of all finite structures, but is restricted to those structures which have a linear order as one of their relations. Thus, for instance, Immerman [15] and Vardi [24] independently showed that the extension FP of first order logic by means of a fixpoint operator characterizes the class PTIME on the class of ordered structures. No such logical charactrerization of PTIME is known for arbitrary finite structures. Similarly, by results of Vardi

* Supported by EPSRC grant GR/H 81108.

[24] and Abiteboul and Vianu [2] it is known that the extension PFP of first order logic by a partial fixpoint operator characterizes the class PSPACE on the class of ordered structures.

In general, FP is strictly weaker than PTIME. That is to say, while every property expressible in FP is decidable in PTIME, there are PTIME properties that are not expressible in FP. The same holds true of PFP and PSPACE. Nevertheless, Abiteboul and Vianu [3] were able to show that FP = PFP, if and only if, PTIME = PSPACE. Thus, even though we do not have a logical characterization of the class PTIME over all finite structures, the open complexity theoretic question about the separation of PTIME and PSPACE can be translated to an equivalent question about the expressive power of two logics on the class of all finite structures. Extending this work, Abiteboul *et al.* [1] defined a variety of fixpoint logics and showed that for a range of complexity classes between PTIME and EXPTIME, open questions about the separations of these classes are equivalent to separations of corresponding fixpoint logics. They also gave characterizations of these fixpoint logics in terms of computability on a *relational* machine model of computation, establishing a general result showing that inclusion relations among relational complexity classes mirror those among the usual computational complexity classes.

The interest in fixpoint logics has also focused attention on the infinitary logic with finitely many variables – $L^{\omega}_{\infty\omega}$. All of the fixpoint logics mentioned above can be seen as fragments of $L^{\omega}_{\infty\omega}$. Recently, considerable effort has been devoted to understanding the model theory of $L^{\omega}_{\infty\omega}$ on finite structures (see, for instance [10, 17, 18]). One of the reasons for this is that definability in $L^{\omega}_{\infty\omega}$ has an elegant characterization in terms of two-player pebble games. Indeed, this has been the main tool used so far in establishing inexpressibility results for fixpoint logic. The logic $L^{\omega}_{\infty\omega}$ has also proved a vehicle for introducing important notions from classical model theory such as elementary equivalence and element types in to finite model theory in a meaningful way, by restricting the number of variables. A systematic study of the k-variable elementary equivalance relation $\equiv^k$ was undertaken in [10]. It is felt that the translation of important open questions in complexity theory into questions about fragments of $L^{\omega}_{\infty\omega}$, as in [1] for instance, provides a greater opportunity for the application of model-theoretic techniques to these questions.

In this paper, we continue the study of the model theory of $L^{\omega}_{\infty\omega}$ by defining a restricted version of second-order logic SO^{ω} that is contained within $L^{\omega}_{\infty\omega}$. This is obtained by restricting the interpretation of second order quantifiers to relations closed under the equivalence relation $\equiv^k$, for some k. We show that the existential fragment of this logic is the class relational NP, while SO^{ω} itself coincides with relational PH. This establishes results in the style of [3] for all levels of the polynomial time hierarchy. Moreover, these are of a somewhat different character to the results in [1] in that the chatacterizations are not in terms of fixpoint logics. We also discuss the expressibility of NP-complete problems in our restricted second order logic, giving examples of natural problems that can be expressed in this way, as well as illustrating techniques for establishing lower bounds by showing, for instance, that 3-colourability cannot be expressed in $L^{\omega}_{\infty\omega}$.

2 Background and Notation

In this section, we fix our notation and examine the necessary background material. We assume familiarity with the basic notions of predicate logic, as well as basic definitions from complexity theory.

A *signature* σ is a finite sequence of relation symbols $(R_1, \ldots, R_s)$, with associated arities $a_1, \ldots, a_s$. A σ-structure $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_s^{\mathfrak{A}})$ consists of a *finite* set A, referred to as the *universe* or *domain* of $\mathfrak{A}$ and interpretations of the relation symbols in σ as relations on A, i.e., $R_i^{\mathfrak{A}} \subseteq A^{a_i}$.

An m-ary *query* q is a map that takes structures over some fixed signature σ to m-ary relations on the domains of the structures and is closed under isomorphisms. That is, for any m-tuple t in a structure $\mathfrak{A}$ and any isomorphism f from $\mathfrak{A}$ to $\mathfrak{B}$, $t \in q(\mathfrak{A})$ if, and only if, $f(t) \in q(\mathfrak{B})$. For instance, a first order formula with m free variables defines an m-ary query. A 0-ary query, also called a *Boolean* query, is a map from the class of σ-structures to the set $\{\text{True}, \text{False}\}$, and can be identified with an isomorphism closed class of σ-structures. In general, we say that a query is expressible in a logic L if there is some formula of L that defines it. By abuse of notation, we will also use L to denote the class of queries that are definable in the logic L. When we speak of the computational complexity of a query q, we mean the complexity of deciding, given a structure $\mathfrak{A}$ and a tuple t from the domain of $\mathfrak{A}$, whether it is the case that $t \in q(\mathfrak{A})$. The measure of the size of the input is the cardinality of the domain of $\mathfrak{A}$.

We say that a logic L *captures* a complexity class C if every query that is expressible by a formula of L is in the complexity class C and conversely, every query that is in C is expressible by a formula of L. We also say that L captures C on a class of structures S when the equivalence between L and C holds for queries whose domain is restricted to S. Note that this usage of the term "capture" is not the same as in [1].

We write Σ^1_1 for the collection of second order sentences in prenex normal form, in which all second order quantifiers precede the first order quantifiers, and which contain only existential second order quantifiers. Fagin [11] proved that Σ^1_1 captures NP. This result was extended by Stockmeyer [23] to show that second order logic captures the polynomial time hierarchy. Indeed, the correspondence between second order logic and the polynomial time hierarchy holds level by level. That is, if Σ^1_{n+1} denotes the collection of sentences of second order logic containing n alternations of second order quantifiers starting with an existential quantifier, then Σ^1_n captures Σ^p_n, the nth level of the polynomial time hierarchy.

2.1 Fixpoint Logics

Let φ be a formula with free individual variables among $x_1, \ldots, x_m$, in the signature σ extended with an additional m-ary predicate symbol R. On σ-structures, φ defines an operator mapping m-ary relations to m-ary relations.

Thus, given a σ-structure $\mathfrak{A}$ and an m-ary relation P in $\mathfrak{A}$, we define $\varphi^{(\mathfrak{A},P)}$ to be $\{s \mid (\mathfrak{A},P) \models \varphi[s]\}$. If this operator is monotone, that is, for every P and Q such that $P \subseteq Q$, $\varphi^{(\mathfrak{A},P)} \subseteq \varphi^{(\mathfrak{A},Q)}$ then it has a least fixed point. While monotonicity is a semantic property, there is a syntactic condition on φ that guarantees that the corresponding operator is monotone. Namely, if φ is R-positive, that is all occurrences of R in φ are in the scope of an even number of negations, then the operator defined by φ is monotone. We write LFP for the closure of first order logic under the operation of taking least fixed points of positive formulas. Immerman [15] and Vardi [24] independently showed that LFP captures the complexity class PTIME over the class of structures which include a linear order as one of their relations.

If φ defines a monotone operator, then its least fixed point in a structure $\mathfrak{A}$ can be obtained by iterating the operator as follows. Define φ^0 to be the empty relation $\emptyset$, and define φ^{i+1} to be $\varphi^{(\mathfrak{A},\varphi^i)}$. Because the operator is monotone, this sequence of relations is increasing, and if $\mathfrak{A}$ has cardinality n, then for some $i \leq n^m$, $\varphi^i = \varphi^{i+1}$. This φ^i is then the least fixed point of φ. A similar iteration can be defined even when φ does not define a monotone operator by taking at each stage the union with the previous stage. That is, define $\varphi^{i+1} = \varphi^i \cup \varphi^{(\mathfrak{A},\varphi^i)}$. The resulting sequence of relations is increasing for any φ, and once again reaches a fixed point for some $i \leq n^m$. This is the *inflationary fixed point* of φ. IFP is defined to be the closure of first order logic under the operation of taking inflationary fixed points of arbitrary formulas. Clearly, for positive formulas, the least fixed point and the inflationary fixed point coincide. Moreover, Gurevich and Shelah [13] showed that for every formula φ, the inflationary fixed point of φ is definable by a formula of LFP. It follows that the two logics IFP and LFP are equivalent on finite structures. For the rest of this paper, we will use the notation FP to denote the logic IFP.

Immerman [15] established a normal form for formulas of LFP. This, along with the result of Gurevich and Shelah mentioned above provides similar normal forms for IFP (see also [2]):

Theorem 1. For every formula φ of FP *there is a formula ψ, which is the inflationary fixed point of a first order formula, such that φ is equivalent to $\exists\bar{x}\psi$ and $\forall\bar{x}\psi$.*

Indeed, we can even require that the first order formula of which ψ is the fixpoint is itself existential (see [2]).

Consider now an arbitrary formula φ that does not necessarily define a monotone operator. The sequence of stages defined by taking $\varphi^0 = \emptyset$ and $\varphi^{i+1} = \varphi^{(\mathfrak{A},\varphi^i)}$ is not necessarily increasing and may or may not converge to a fixed point. However, if there is an i such that $\varphi^i = \varphi^{i+1}$, then there is such an $i \leq 2^{n^m}$. The *partial fixed point* of φ is defined to be φ^i for i such that $\varphi^i = \varphi^{i+1}$, if such an i exists, and empty otherwise. PFP denotes the closure of first order logic under an operation defining the partial fixed point of formulas. Abiteboul and Vianu [2] showed that PFP is equivalent to the language *while* introduced by Chandra and Harel [6]. Vardi [24] showed that the language *while* captures the class PSPACE on the collection of structures with a linear order.

There is an apparently more general form of inductive definition, where a query is defined by simultaneous induction of a number of formulas. Let $S_0, \dots, S_l$ be a sequence of relation symbols that do not occur in the signature σ, with associated arities $a_0, \dots, a_l$. Further, let $\varphi_0, \dots, \varphi_l$ be a sequence of formulas in the signature formed by extending σ by $S_0, \dots, S_l$, where φ_j defines a query of arity a_j. We define the stages of the simultaneous induction of the sequence $(\varphi_0, \dots, \varphi_l)$ by:

$$\begin{aligned} \varphi_j^0 &= \emptyset \\ \varphi_j^{i+1} &= \varphi_j^{(\mathfrak{A}, \varphi_0^i, \dots, \varphi_l^i)} (\text{or } \varphi_j^i \cup \varphi_j^{(\mathfrak{A}, \varphi_0^i, \dots, \varphi_l^i)} \text{ for the inflationary case}). \end{aligned}$$

The sequence reaches a fixed point if there is an i such that $\varphi_j^i = \varphi_j^{i+1}$ for all $0 \leq j \leq l$, and the relation defined by the fixpoint is then φ_0^i.

Moschovakis [21] showed that allowing simultaneous inductions does not increase the expressive power of fixpoint logics (see also [19] for a discussion and [2] for the case of partial fixpoints):

Theorem 2. Every query defined by a simultaneous inflationary (resp. partial) induction is definable in FP *(resp.* PFP *).*

In the light of Theorem 2, we will use simultaneous inductions in this paper wherever it makes the exposition clearer.

Abiteboul *et al.* [1] extended the above framework of fixpoint logics by defining a range of fixpoint logics obtained by varying two parameters – the control mechanism and the semantics of the fixpoint iteration. The control mechanism can be deterministic, non-deterministic or alternating, and the semantics can be inflationary or non-inflationary. The two fixpoint logics considered above, FP and PFP, in this terminology, are both deterministic in their control, with inflationary and non-inflationary semantics, respectively. We will now consider the non-deterministic inflationary fixpoint logic, introduced by Abiteboul *et al.* We denote this logic NFP, for non-deterministic fixpoint logic.[2]

Given two formulas φ_0 and φ_1 in a signature σ extended by an additional m-ary predicate R, we define in any σ-structure $\mathfrak{A}$, a sequence of stages of the pair (φ_0, φ_1) indexed by binary strings:

$$\begin{aligned} \varphi^\epsilon &= \emptyset, \text{ for the empty string } \epsilon \\ \varphi^{s \cdot 0} &= \varphi^s \cup \varphi_0^{(\mathfrak{A}, \varphi^s)} \\ \varphi^{s \cdot 1} &= \varphi^s \cup \varphi_1^{(\mathfrak{A}, \varphi^s)} \end{aligned}$$

We now define the non-deteministic fixed point of the pair (φ_0, φ_1) in the structure $\mathfrak{A}$ to be $\bigcup_{s \in \{0,1\}^*} \varphi^s$. The logic NFP is the closure of first order logic under the operation of taking non-deterministic fixed points, with the proviso that the fixpoint operator does not occur within the scope of a negation.

We observe, without proof, that Theorems 1 and 2 extend directly to NFP as well. That is, we define a simultaneous non-deterministic induction by a sequence

[2] This notation is different from [1], where NFP denotes the logic we call PFP.

of pairs of formulas: $\varphi_i = (\varphi_{i,0}, \varphi_{i,1}), 0 \leq i \leq l$ in a signature σ extended with new relation symbols $S_0, \ldots, S_l$. The stages of this induction on a σ-structure $\mathfrak{A}$ are defined as follows:

$$\begin{aligned}
&\varphi_i^{\epsilon} \ = \emptyset, \text{ for the empty string } \epsilon \\
&\varphi_i^{s \cdot 0} = \varphi^s \cup \varphi_{i,0}^{(\mathfrak{A}, \varphi_0^s \ldots \varphi_l^s)} \\
&\varphi_i^{s \cdot 1} = \varphi^s \cup \varphi_{i,1}^{(\mathfrak{A}, \varphi_0^s \ldots \varphi_l^s)}
\end{aligned}$$

The non-deteministic fixed point of the sequence is given by $\bigcup_{s \in \{0,1\}^*} \varphi_0^s$. The following lemma is proved in the same way as Theorem 2.

Lemma 3. Every query defined by a simultaneous non-deterministic induction is definable in NFP.

Similarly, the proof of the normal form result, Theorem 1, can also be extended to NFP.

Lemma 4. Every formula of NFP *is equivalent to a formula of the form $\exists \bar{x} \varphi$, where φ is the non-deterministic fixed point of a pair of first order formulas.*

2.2 Infinitary Logic

The infinitary logic $L_{\infty\omega}$ is obtained by closing first order logic under conjunctions and disjunctions of arbitrary (not just finite) sets of formulas. This logic is of little use in the study of finite models, since every query on the class of finite structures is expressible in $L_{\infty\omega}$. However, the restriction of $L_{\infty\omega}$ where we only allow finitely many variables to appear in any given formula, has proved to be of great value in studying the expressive power of fixed point logics on finite structures.

More formally, let $L^k_{\infty\omega}$ denote the class of formulas of $L_{\infty\omega}$ in which all variables (free or bound) are among $x_1, \ldots, x_k$. Also, let $L^\omega_{\infty\omega} = \bigcup_{k \in \omega} L^k_{\infty\omega}$. The logic $L^\omega_{\infty\omega}$ was introduced by Barwise [4] in order to study inductive definitions on infinite structures. It was shown by Rubin [22] that for a fixed infinite structure, the least fixed point of any first order operator is expressible in $L^\omega_{\infty\omega}$. A similar result was obtained for the class of all finite structures by Kolaitis and Vardi [18], who showed that in this case, both FP and PFP can be seen as fragments of $L^\omega_{\infty\omega}$. This also applies to NFP, giving us the following picture:

$$\text{FP} \subseteq \text{NFP} \subseteq \text{PFP} \subseteq L^\omega_{\infty\omega}.$$

The last containment in the above is a proper one, since Kolaitis and Vardi [18] show that there are non-recursive queries that can be expressed in $L^\omega_{\infty\omega}$, while every query definable in PFP is computable in PSPACE. Indeed one can show that just as $L_{\infty\omega}$ is complete in its expressive power, so $L^\omega_{\infty\omega}$ is complete on ordered structures (for a fuller discussion of this, see [8]):

Proposition 5. For every signature σ, there is a $k \in \omega$ such that every query on ordered σ-structures is expressible in $L^k_{\infty\omega}$.

We write L^k for the first order fragment of $L^k_{\infty\omega}$, i.e. the formulas of first order logic that contain only the variables $x_1, \ldots, x_k$.

Recall that for a structure $\mathfrak{A}$ and a tuple s of elements of $\mathfrak{A}$, the first order type of s in $\mathfrak{A}$, denoted Type($\mathfrak{A}, s$) is the set of formulas φ such that $\mathfrak{A} \models \varphi[s]$. The following variant of this notion was introduced in [10], and has proved to be very useful in studying expressibility in $L^\omega_{\infty\omega}$.

Definition 6. For any structure $\mathfrak{A}$ and a tuple s of elements of $\mathfrak{A}$, $\text{Type}^k(\mathfrak{A}, s)$ denotes the set of formulas φ of L^k such that $\mathfrak{A} \models \varphi[s]$.

We write $(\mathfrak{A}, s) \equiv^k (\mathfrak{B}, t)$ to denote $\text{Type}^k(\mathfrak{A}, s) = \text{Type}^k(\mathfrak{B}, t)$. We also write k-size($\mathfrak{A}$) to denote the number of equivalence classes of the relation $\equiv^k$ in the structure $\mathfrak{A}$.

Kolaitis and Vardi [18] showed that the equivalence relation $\equiv^k$ coincides, on finite structures, with the apparently stronger notion of equivalence in $L^k_{\infty\omega}$. That is, they showed that if $\mathfrak{A}$ and $\mathfrak{B}$ are finite structures and $(\mathfrak{A}, s) \equiv^k (\mathfrak{B}, t)$ then for every formula $\varphi \in L^k_{\infty\omega}$, $\mathfrak{A} \models \varphi[s]$ if and only if $\mathfrak{B} \models \varphi[t]$. Kolaitis and Vardi [18] also showed that a query q is definable in $L^k_{\infty\omega}$ if, and only if, it is closed under the relation $\equiv^k$, i.e. if $s \in q(\mathfrak{A})$ and $(\mathfrak{A}, s) \equiv^k (\mathfrak{B}, t)$ then $t \in q(\mathfrak{B})$. It was shown in [10] that if $\mathfrak{A}$ is a finite structure, there is a formula $\varphi \in \text{Type}^k(\mathfrak{A}, s)$ such that for any structure $\mathfrak{B}$, $\mathfrak{B} \models \varphi[t]$ if and only if $(\mathfrak{A}, s) \equiv^k (\mathfrak{B}, t)$.

The equivalence relation $\equiv^k$ has an elegant characterization in terms of Ehrenfeucht-Fraïssé style pebble games, essentially given by Barwise [4] (see also [14]). The game board consists of two structures $\mathfrak{A}$ and $\mathfrak{B}$ and a supply of k pairs of pebbles $(a_i, b_i), 1 \leq i \leq k$. The pebbles $a_1, \ldots, a_l$ are initially placed on the elements of an l-tuple s in $\mathfrak{A}$, and the pebbles $b_1, \ldots, b_l$ on a tuple t in $\mathfrak{B}$. There are two players, Spoiler and Duplicator. At each move of the game, Spoiler picks up a pebble (either an unused pebble or one that is already on the board) and places it on an element of the corresponding structure. For instance he might take pebble b_i and place it on an element of $\mathfrak{B}$. Duplicator must respond by placing the other pebble of the pair in the other structure. In the above example, she must place a_i on an element of $\mathfrak{A}$. If at the end of the move the partial map $f : \mathfrak{A} \to \mathfrak{B}$ given by $a_i \mapsto b_i$ is not a partial isomorphism, then Spoiler has won the game, otherwise it can continue for another move. Duplicator has a strategy to avoid losing for n moves, starting with the initial position $(\mathfrak{A}, s)$ and $(\mathfrak{B}, t)$ if, and only if, $(\mathfrak{A}, s)$ and $(\mathfrak{B}, t)$ cannot be distinguished by any formula of L^k of quantifier rank n or less. Hence, if Duplicator has a strategy to play the game indefinitely without losing, then $(\mathfrak{A}, s) \equiv^k (\mathfrak{B}, t)$.

The relation $\equiv^k$ is itself uniformly definable in FP [10, 17]:

Theorem 7. There is a formula η of FP, *with $2k$ free variables, such that on any finite structure $\mathfrak{A}$, given two k-tuples s and t in $\mathfrak{A}$, $\mathfrak{A} \models \eta[s, t]$ if, and only if, $(\mathfrak{A}, s) \equiv^k (\mathfrak{A}, t)$.*

Moreover, we can also write a formula λ of FP which uniformly orders $\equiv^k$ equivalence classes (see [3, 10]). That is, on any finite structure $\mathfrak{A}$, λ defines a pre-order such that the corresponding equivalence relation is $\equiv^k$:

Theorem 8. There is a formula λ of FP, *with $2k$ free variables, such that on any finite structure $\mathfrak{A}$, λ defines a reflexive and transitive relation $\leq^k$ on k-tuples such that for every two k-tuples s and t, either $s \leq^k t$ or $t \leq^k s$ and, both $s \leq^k t$ and $t \leq^k s$ hold if and only if $(\mathfrak{A}, s) \equiv^k (\mathfrak{A}, t)$.*

Thus, λ can be seen as defining a total order on the equivalence classes of $\equiv^k$. The FP definition of this order allows us to define an FP reduction which maps any structure $\mathfrak{A}$ to a quotient structure $\mathfrak{A}/\equiv^k$ which is linearly ordered. Using such a reduction, Abiteboul and Vianu [3] showed that FP = PFP if and only if PTIME = PSPACE (see also the exposition in [10]).

Abiteboul *et al.* [1] extend this by showing that the logic NFP captures the relational complexity class NP_r, whereby it follows that:

$$\mathrm{FP} = \mathrm{NFP} \quad \text{if and only if} \quad \mathrm{PTIME} = \mathrm{NP}; \text{ and}$$

$$\mathrm{NFP} = \mathrm{PFP} \quad \text{if and only if} \quad \mathrm{NP} = \mathrm{PSPACE}.$$

3 A Restricted Second Order Logic

The fixpoint logics can be viewed as effective fragments of $L^{\omega}_{\infty\omega}$, as we saw in Section 2.2. In this section, we explore a different way of obtaining an effective fragment of $L^{\omega}_{\infty\omega}$, by a restricted form of second order quantification. This provides a logical characterization of some relational complexity classes that is not based on a fixpoint logic, and is closer in spirit to Fagin's characterization of NP.

Definition 9.

- For an l-ary relation symbol R, and $k \geq l$, we define the second order quantifier $\exists^k R$ to have the following semantics: $\mathfrak{A} \models \exists^k R \varphi$ if there is an $X \subseteq A^l$ such that X is closed under the equivalence relation $\equiv^k$ in $\mathfrak{A}$, and $(\mathfrak{A}, X) \models \varphi$. As usual, $\forall^k R$ abbreviates $\neg\exists^k R\neg$.
- $\Sigma^{1,\omega}_1$ denotes the class of formulas of the form $\exists^{k_1} R_1 \ldots \exists^{k_m} R_m \varphi$, where φ is first order.
- $\Pi^{1,\omega}_1$ denotes the class of formulas of the form $\forall^{k_1} R_1 \ldots \forall^{k_m} R_m \varphi$, where φ is first order.
- $\Sigma^{1,\omega}_{n+1}$ denotes the class of formulas of the form $\exists^{k_1} R_1 \ldots \exists^{k_m} R_m \varphi$, where φ is $\Pi^{1,\omega}_n$.
- $\Pi^{1,\omega}_{n+1}$ denotes the class of formulas of the form $\forall^{k_1} R_1 \ldots \forall^{k_m} R_m \varphi$, where φ is $\Sigma^{1,\omega}_n$.
- $\mathrm{SO}^{\omega} = \bigcup_{n \in \omega} \Sigma^{1,\omega}_n$.

The logic SO^{ω} is a restricted version of second order logic which forms an effective fragment of $L^{\omega}_{\infty\omega}$. In Theorem 12 below, we will see that it is in fact contained in PFP. We begin by establishing its relationship with the usual second order logic.

Lemma 10. For every n, $\Sigma^{1,\omega}_n \subseteq \Sigma^1_n$ and $\Pi^{1,\omega}_n \subseteq \Pi^1_n$.

Proof. We present the proof for the Σ-classes. The proof for the Π-classes is analogous.

Let $\varphi \in \Sigma_n^{1,\omega}$ be a formula $\exists^{k_1} R_1 \ldots Q^{k_q} R_q \psi$, where Q is $\exists$ or $\forall$, depending on whether n is odd or even. Clearly, the query defined by φ can be expressed as:

$$\exists R_1 \ldots Q R_q (\psi \wedge \bigwedge_{1 \leq i \leq q} \gamma^{k_i}(R_i)), \qquad (1)$$

where $\gamma^{k_i}(R_i)$ asserts that R_i is $\equiv^{k_i}$-closed. By Theorem 7 each γ^k is definable in FP. Since FP $\subseteq \Sigma_1^1 \cap \Pi_1^1$, it follows that the query (1) is definable in Σ_n^1.

Theorem 11. *On ordered structures, for every $n \in \omega$, $\Sigma_n^{1,\omega} = \Sigma_n^1$ and $\Pi_n^{1,\omega} = \Pi_n^1$.*

Proof. Since Lemma 10 holds on arbitrary structures, it holds on ordered structures, in particular. Thus, we only need to show the inclusions $\Sigma_n^1 \subseteq \Sigma_n^{1,\omega}$ and $\Pi_n^1 \subseteq \Pi_n^{1,\omega}$.

It follows from Proposition 5 that for every signature σ there is a k_σ such that, if $\mathfrak{A}$ is an ordered σ-structure and R is any l-ary relation on $\mathfrak{A}$ for $l \leq k_\sigma$, then R is $\equiv^{k_\sigma}$-closed. Let $\varphi \in \Sigma_n^1$ be a σ-sentence $\exists R_1 \ldots Q R_q \psi$. For each R_i of arity a_i, let $k_i = \max(a_i, k_\sigma)$. Then it is easily seen that φ is equivalent, on ordered structures, to the sentence

$$\exists^{k_1} R_1 \ldots Q^{k_q} R_q \psi.$$

The proof for Π_n^1 sentences is similar.

Theorem 11 establishes that the restricted second order logic SO$^\omega$ is not really restricted on ordered structures. In what follows, we establish the relationship of SO$^\omega$ to the fixpoint logics.

Theorem 12. SO$^\omega$ $\subseteq$ PFP.

Proof. It suffices to show that, given a formula ψ of PFP, there is a formula φ of PFP equivalent to $\exists^k R\psi$.

Any relation P that is $\equiv^k$-closed on a structure $\mathfrak{A}$ can be seen as a set of $\equiv^k$ equivalence classes. Thus, the pre-order $\preceq^k$ of Theorem 8, being a linear order on the collection of $\equiv^k$ equivalence classes, induces a lexicographical ordering of all $\equiv^k$-closed relations on $\mathfrak{A}$. Moreover, using the FP formula defining $\preceq^k$, we can write an FP formula $\nu(P)$ which defines, for any $\equiv^k$-closed relation P of arity l, the lexicographically next such relation.

We assume that the formula $\exists^k R\psi$ has free individual variables among $x_1, \ldots, x_m$ and therefore defines an m-ary query. We define this query by means of a simultaneous induction of two formulas. We therefore have two induction variables S and R of arity m and l, repectively. At succesive stages of the induction R takes on, in lexicographical order, the values of $\equiv^k$-closed relations, reaching a fixed point when it contains all l-tuples. At the same time S accumulates the m-tuples that satisfy $\psi(R)$. Formally, we define the formulas φ_S and φ_R as follows:

$$\varphi_S \equiv S(\bar{x}) \vee \psi(R)$$
$$\varphi_R \equiv \forall \bar{y} R(\bar{y}) \vee (\exists \bar{y} \neg R(\bar{y}) \wedge \nu(R)).$$

It can be verified that the simultaneous partial fixed point of this induction yields the query $\exists^k R\psi$. Therefore, by Theorem 2, we have a formula of PFP that expresses this query.

We now show that the existential fragment of SO^ω is, in fact, equivalent to NFP. In the next two lemmas, we state the crucial results for proving the two directions of the equivalence of $\Sigma_1^{1,\omega}$ and NFP.

Lemma 13. For any pair of first order formulas φ_0 and φ_1, there is a formula of $\Sigma_1^{1,\omega}$ that defines the non-deterministic fixed point of (φ_0, φ_1).

Proof. We note first that by the inflationary nature of the non-deterministic fixpoint operator, if s_1 is a prefix of s_2, then $\varphi^{s_1} \subseteq \varphi^{s_2}$ in any structure $\mathfrak{A}$. Thus, if we consider any increasing sequence of binary strings, then the corresponding sequence of stages is increasing. It follows that, if we let k be the maximum of the number of distinct variables in φ_0 and φ_1, then for binary strings s such that length$(s) \geq k$-size$(\mathfrak{A})$, $\varphi^s = \varphi^{s \cdot 0} = \varphi^{s \cdot 1}$. Consider the formula:

$$\exists^{2k} O \exists^{k+m} R\psi$$

with free individual variables $x_1, \ldots, x_m$, where ψ asserts that:

- O is a pre-order of k-tuples;
- $R^0 = \emptyset$ and for every i, either $R^{i+1} = R^i \cup \varphi_0^{R^i}$ or $R^{i+1} = R^i \cup \varphi_1^{R^i}$, where R^i is:

  $\{t \mid R(s,t)$ for some s in the ith equivalence class determined by the pre-order $O\}$;

  and
- $R^m(x_1, \ldots, x_m)$, where m is the length of the pre-order O.

It can be verified that this formula expresses the non-deterministic fixpoint of the pair (φ_0, φ_1).

In the other direction, we have the following lemma.

Lemma 14. If ψ is a formula of NFP, *then there is a formula of* NFP *that is equivalent to $\exists^k R\psi$.*

Proof. For simplicity, we assume that the arity of R is k and that the free individual variables in ψ are among $x_1, \ldots, x_m$.

As in the proof of Theorem 12, we are going to use the FP definition of the order $\leq^k$ of $\equiv^k$ equivalence classes. Intuitively, we want to define an induction that steps along this order and at each stage decides non-deterministically whether or not to include the current equivalence class in the relation R. For this, we essentially need to maintain three relations: one, S, to count the equivalence classes that have been visited, one to include those equivalence classes that have been chosen to be in R, and finally one, P, to construct the relation defined by ψ, given the candidate R. We do this by a simultaneous induction of three pairs of formulas, $(\varphi_{P,0}, \varphi_{P,1}), (\varphi_{R,0}, \varphi_{R,1})$ and $(\varphi_{S,0}, \varphi_{S,1})$, with relation symbols R and S of arity k and P of arity m. Note that in the definition below, $\varphi_{P,0} \equiv \varphi_{P,1}$ and $\varphi_{S,0} \equiv \varphi_{S,1}$ so the non-determinism is confined to the pair $(\varphi_{R,0}, \varphi_{R,1})$.

$$\begin{aligned}
&\varphi_{P,0} \equiv \varphi_{P,1} \equiv \forall \bar{y} S(\bar{y}) \land \psi(R) \\
&\varphi_{S,0} \equiv \varphi_{S,1} \equiv \forall \bar{y}(\lambda(\bar{y}, \bar{x}) \rightarrow S(\bar{y}))
\end{aligned}$$

$$\begin{aligned}
&\varphi_{R,0} \equiv x \neq x \\
&\varphi_{R,1} \equiv \forall \bar{y}(\lambda(\bar{y}, \bar{x}) \rightarrow S(\bar{y})).
\end{aligned}$$

In the above λ is the FP formula in Theorem 8 that defines the pre-order $\preceq^k$. It is clear that any inflationary fixpoint can be expressed as a non-deterministic fixpoint (simply by taking $\varphi_0 = \varphi_1$). A slight complication arises because in the above formulas λ appears within the scope of a negation symbol. However, by Theorem 1, we know that the negation of an FP formula can be expressed without the fixpoint operator appearing inside the scope of a negation.

It can be verified that the simultaneous non-deterministic fixpoint of the above system defines the query $\exists^k R\psi$. Therefore, by Lemma 3, there is a formula of NFP that defines the same query.

The following theorem is immediate from Lemmas 4, 13, and 14.

Theorem 15. $\Sigma_1^{1,\omega} = \mathrm{NFP}$.

Remark. The proof of Theorem 15 can be extended to show that, if we close the logic NFP simultaneously under negation and the operation of taking non-deterministic fixpoints, we obtain a logic equivalent to SO^ω. Moreover, the alternations of negations and fixpoints correspond exactly to the second order quantifier alternations in SO^ω. Similarly, Abiteboul *et al.* [1] also define an alternating inflationary fixpoint logic, which they show to be equivalent to PFP. One can show that the fragment of this logic obtained by allowing only a bounded number of alternations is equivalent to SO^ω. Once again, the number of alternations corresponds exactly to the number of alternations of second order quantifiers in SO^ω.

The following corollaries follow immediately from Theorem 15.

Corollary 16. $\mathrm{FP} \subseteq \Sigma_1^{1,\omega} \cap \Pi_1^{1,\omega}$.

Corollary 17. $\mathrm{FP} = \Sigma_1^{1,\omega}$ *if and only if* PTIME = NP.

Corollary 18. $\Sigma_1^{1,\omega} = \mathrm{PFP}$ *if and only if* NP = PSPACE.

Moreover, an application of the same methods also gives us:

Theorem 19. $\mathrm{SO}^\omega = \mathrm{PFP}$ *if and only if* PH = PSPACE.

By Corollaries 17, and 18 and Theorem 19, the inclusion relations between SO^ω and the fixpoint logics FP and PFP are equivalent to open problems in complexity theory. The next result shows that this is also true of the levels of the hierarchy within SO^ω.

Theorem 20. For every $i, j \in \omega$, $\Sigma_i^{1,\omega} = \Sigma_j^{1,\omega}$ if and only if $\Sigma_i^p = \Sigma_j^p$; and $\Sigma_i^{1,\omega} = \Pi_i^{1,\omega}$ if and only if $\Sigma_i^p = \Pi_i^p$.

Finally, we also observe that when a sentence φ of $\Sigma_i^{1,\omega}$ is translated to the ordered quotient structure $\mathfrak{A}/\equiv^k$, the resulting sentence is, in fact, in the monadic fragment of Σ_1^1, i.e. it only uses quantification over sets. Writing, $\mathrm{mon}.\Sigma_i^1$ for the monadic fragment of Σ_i^1, we can then extract the following result from the proof of Theorem 20. Note that this result is not about the logic SO^ω. It is a result about the unrestricted second order logic, although it is obtained by using facts about SO^ω in the proof.

Theorem 21. For any $i, j \in \omega$, if $\mathrm{mon}.\Sigma_i^1 = \mathrm{mon}.\Sigma_j^1$ on ordered strucutures, then $\Sigma_i^p = \Sigma_j^p$.

4 NP-Complete Problems

While the logic FP cannot express some easily computable properties – such as the property of a graph having even cardinality, which is not even expressible in $L_{\infty\omega}^\omega$ – it can nevertheless express some P-complete problems such as the path systems problem and alternating transitive closure (see [18]). Similarly, Abiteboul *et al.* [1] show that there are natural PSPACE-complete problems that can be expressed in PFP. In this section, we examine the expressibility of NP-complete problems in the logic $\Sigma_1^{1,\omega}$.

In one sense, it is easy to see that there are NP-complete problems that can be expressed in $\Sigma_1^{1,\omega}$, since this logic captures NP on ordered structures (see Theorem 11). Thus, if we take any NP-complete problem and consider the set of its instances with linear order, we obtain a problem that is still NP-complete and is expressible in $\Sigma_1^{1,\omega}$. For instance, consider the class of structures $(V, E, \leq)$, where $\leq$ is a linear order on V, and the graph (V, E) is Hamiltonian.

However, in the absence of linear order, many natural NP-complete problems cannot be expressed in $L_{\infty\omega}^\omega$ and *a fortiori* not in $\Sigma_1^{1,\omega}$. Immerman [14] showed, essentially, that Hamiltonicity and clique are not definable in $L_{\infty\omega}^\omega$. We present, as an example, a simple proof that Hamiltonicity is not in $L_{\infty\omega}^\omega$.

Example 1. Consider the complete bipartite graph $K_{m,n}$. It is easily verified that this graph is Hamiltonian if and only if $m = n$. An easy pebble game argument shows that $K_{k,k} \equiv^k K_{k,k+1}$. Since $K_{k,k}$ is Hamiltonian and $K_{k,k+1}$ is not, it follows that Hamiltonicity is not in $L_{\infty\omega}^k$ for any k.

Lovász and Gács [20] show that the problem of propositional satisfiability (SAT) is complete for NP even under first order reductions. Since $L_{\infty\omega}^\omega$ is closed under first order reductions, it follows that SAT is not definable in $L_{\infty\omega}^\omega$, for otherwise

NP would be contained in $L^{\omega}_{\infty\omega}$, which we know is not the case. Similarly, Dahlhaus [7] shows that both Hamiltonicity and clique are also NP-complete under first order reductions and this provides an alternative proof that these problems are not expressible in $L^{\omega}_{\infty\omega}$. Essentially, it is a consequence of the completeness results that we can take the proof that some query in NP is not in $L^{\omega}_{\infty\omega}$ – say the even cardinality query – and translate it into a proof that Hamiltonicity or clique is not in $L^{\omega}_{\infty\omega}$.

In contrast, 3-colourability is an NP-complete problem that is known not to be complete with respect to first order reductions. By a result of [9], we know that the class of queries that are reducible to 3-colourability obeys a 0-1 law. That is, for every Boolean query in this class, the proportion of structures of size n that are instances of the query tends to either 0 or 1 as n goes to infinity. It follows that straightforward counting arguments such as in Example 1 will not suffice to show that 3-colourability is not expressible in $L^{\omega}_{\infty\omega}$. By taking a different approach, we are nevertheless able to show below that 3-colourability is not definable in $L^{\omega}_{\infty\omega}$. This answers an open question posed by Kolaitis and Vardi [18].

4.1 3-Colourability

In order to show that 3-colourability is not expressible in $L^{\omega}_{\infty\omega}$, we adapt a construction due to Cai *et al.* [5] which shows that there is a PTIME query that is not expressible in the extension of FP by counting.

The crucial idea in the construction of Cai *et al.* is to construct graphs X_d, which include d distinguished pairs of nodes $(a_1, b_1), \ldots, (a_d, b_d)$ with the following property:

(*) for every subset S of $\{1, \ldots, d\}$ which is of even cardinality, there is an automorphism of X_d which exchanges a_i and b_i for $i \in S$, while fixing a_i and b_i for $i \notin S$. There is no automorphism of X_d that does this for a set S of odd cardinality.

We refer to the pair of points (a_i, b_i) as the ith gate of X_d.

We can construct such an $X_d$by including, in addition to the d gates, 2^{d-1} nodes v_S, one for each even sized subset S of $\{1, \ldots, d\}$. The graph X_d then contains the edges (a_i, v_S) for $i \in S$ and (b_i, v_S) for $i \notin S$. It can be easily verified that X_d has property (*).

Example 2. The graph X_3 is depicted in Figure 1.

Let G be a graph such that every vertex in G has degree at least 2. The graph $X(G)$ is defined as follows. Every vertex v in G is replaced by a copy of X_d, where d is the degree of v, with each edge incident on v being assigned a gate of X_d. We denote the copy of X_d that replaces v by X^v, and its ith gate by (a^v_i, b^v_i). For an edge (u, v) of G, let the gates in X^u and X^v assigned to this edge be (a^u_i, b^u_i) and (a^v_j, b^v_j). The graph $X(G)$ contains the two edges (a^u_i, a^v_j) and (b^u_i, b^v_j). We also define the graph $\tilde{X}(G)$, which is obtained from

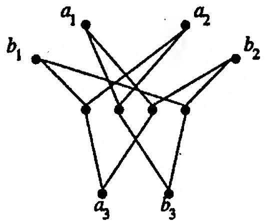


Fig. 1. X_3

$X(G)$ by "twisting" exactly one edge. That is, for one edge (u, v) of G, in place of the edges (a_i^u, a_j^v) and (b_i^u, b_j^v), we include (a_i^u, b_j^v) and (b_i^u, a_j^v).

We now state two lemmas regarding this construction due to Cai *et al.* [5].

Lemma 22. *$X(G)$ and $\tilde{X}(G)$ are not isomorphic.*

We omit a full proof of this lemma, oberving only that any isomorphism from $X(G)$ to $\tilde{X}(G)$ would, restricted to some X^v, yield an automorphism of X^v which exchanges a_i and b_i for an odd number of gates. This, however, would violate the property (*) of X^v.

Recall that a separator of a graph $G = (V, E)$ is a set of nodes $U \subseteq V$, such that the subgraph of G induced by $V \setminus U$ has no connected component containing more than $|V|/2$ nodes. This allows us to formulate the second lemma due to Cai *et al.* [5].

Lemma 23. *If G has no separator of cardinality k, then $X(G) \equiv^k \tilde{X}(G)$.*

Once again, we omit a detailed proof of this lemma, and present instead an informal description of Duplicator's strategy in the pebble game. At any stage in the pebble game, there are at most k pebbles on each of $X(G)$ and $\tilde{X}(G)$. Consider the graphs formed by removing from $X(G)$ and $\tilde{X}(G)$ any X^v that contains a pebbled vertex. Since G has no separator of cardinality k, the resuling graphs each contain a connected component that includes more than half of all the vertices. Duplicator's strategy is essentially to "hide the twist" in this large component. Clearly, the only way the Spoiler can win the game is by isolating the twist, i.e., by placing pebbles on two of the four vertices in the two gates of the twisted edge (u, v), in such a way as to force Duplicator to interchange the vertices in one of the gates, say of X^u. She is then forced to interchange the vertices in another of the gates of X^u, effectively moving the twist to another location. Since after every move there is an unpebbled component containing more than half the vertices, these components must overlap from one move to the next. This allows Duplicator to always keep the twisted edge in the large component, and therefore Spoiler cannot isolate it.

To adapt the construction of Cai *et al.* to show that 3-colourability is not definable in $L^{\omega}_{\infty\omega}$, we construct a gadget C_d for every d, along the lines of

X_d above, that has some additional properties. We state the relevant properties here, and defer the explicit construction to a later point in this section.

1. C_d contains d gates, each consisting of three nodes (a_i, b_i, c_i), and these nodes are connected by edges to form a triangle.
2. For every subset S of $\{1, \ldots, d\}$ of even size, there is an automorphism of C_d that exchanges a_i and b_i for $i \in S$, while fixing nodes in all other gates. There is no such automorphism for odd size sets S.
3. C_d is 3-colourable, and in any valid 3-colouring of C_d all c_i are assigned the same colour. There is a valid 3-colouring of C_d in which all a_i are assigned the same colour. Finally, this 3-colouring is unique up to renaming of colours and automorphisms of C_d.

One consequence of these conditions is that if d is even, then in any 3-colouring of C_d and for any of the colours, the number of a_i that are assigned that colour must be even.

We now define, for every graph G, the graphs $C(G)$ and $\tilde{C}(G)$ along the lines of $X(G)$ and $\tilde{X}(G)$ above. The only difference is that each edge (u, v) of G is now replaced by three edges in $C(G)$, (a_i^u, a_j^v), (b_i^u, b_j^v) and (c_i^u, c_j^v). The graph $\tilde{C}(G)$ is obtained from $C(G)$ by replacing exactly one pair (a_i^u, a_j^v), (b_i^u, b_j^v) of edges by (a_i^u, b_j^v), (b_i^u, a_j^v). The following lemma is now immediate, along the lines of Lemma 23.

Lemma 24. If G has no separator of cardinality k, then $C(G) \equiv^k \tilde{C}(G)$.

We proceed to construct, for every $k \in \omega$, a graph which has no separator of cardinality k, which is 3-colourable and such that its 3-colouring is unique up to a renaming of colours.

Definition 25. A *triangular mesh* of order n, denoted T_n, is a graph with n^2 vertices: $v_{(i,j)}, 0 \le i, j < n$ and the following edges for each i and j:

$$[v_{(i,j)}, v_{(i+1,j)}]; \quad [v_{(i,j)}, v_{(i,j+1)}]; \quad [v_{(i,j)}, v_{(i+1,j+1)}],$$

where the additions are all modulo n.

We now establish the relevant properties of triangular meshes in the next three lemmas.

Lemma 26. For $n \ge 3$, T_n has no separator of cardinality n.

Proof. For each i, define the row R_i to be the set of vertices $\{v_{(i,j)} \mid 0 \le j < n\}$. Similarly define the column $C_j = \{v_{(i,j)} \mid 0 \le i < n\}$. Note that the subgraph of T_n induced by each row and each column is a cycle of length n.

Let U be any subset of the vertices of T_n such that $|U| = n$. We first note that if U contains one vertex from every row or one vertex from every column, then the result of removing the vertices in U from T_n is a connected graph. To see this, suppose U contains one vertex from every row, then after the removal of vertices in U; every row remains connected, since it is a cycle with one vertex removed, and since two successive rows are connected by $2n$ edges, the removal of one vertex in each row will not disconnect them. Thus such a U does not form a separator.

Next, we consider a set U of cardinality n which for some row and for some column includes at least two of its vertices. But then, there must be R_i and C_j such that $R_i \cap U = \emptyset$ and $C_j \cap U = \emptyset$. Note that $R_i \cup C_j$ contains $2n-1$ vertices. Now, for any other column C_l such that $|C_l \cap U| \leq 1$, all the elements of C_l are connected to $v_{(i,l)}$ by a path that does not include a vertex in U. It follows that, after removing U, at least $n-2$ of the vertices in at least half the remaining columns are in the same connected component as $R_i \cup C_j$. Thus, this component contains at least $(2n-1)+(n-2)^2/2$ vertices, which is more than half the vertices of T_n. Thus, U is not a separator of T_n.

Lemma 27. *If n is a multiple of 3, then T_n is 3-colourable.*

Proof. We define a 3-colouring $\chi : T_n \rightarrow \{0,1,2\}$ given by:

$$\chi(v_{(i,j)}) = l \quad \text{if and only if} \quad i+j \equiv l \pmod 3.$$

It is easily seen that this is a valid 3-colouring of the graph.

Lemma 28. *If n is a multiple of 3, then $C(T_n)$ is 3-colourable, and $\tilde{C}(T_n)$ is not 3-colourable.*

Proof. To see that $C(T_n)$ is 3-colourable, consider a valid 3-colouring of T_n, $\chi : T_n \rightarrow \{0,1,2\}$. For each vertex v of T_n, we can colour the graph C^v in such a way that all c_i^v are assigned the colour $\chi(v)$, all a_v^i are assigned the colour $(\chi(v)+1) \bmod 3$ and all b_v^i are assigned the colour $(\chi(v)+2) \bmod 3$. It can then be easily verified that this results in a valid 3-colouring of $C(T_n)$.

To show that $\tilde{C}(T_n)$ is not 3-colourable, we make the following observations. First, the edges of T_n can be partitioned into $3n$ sets, each of which forms a cycle of length n. These are given by the n rows $R_i = \{(i,j) \mid 0 \leq j < n\}$, the n columns $C_j = \{(i,j) \mid 0 \leq i < n\}$ and the n diagonals $D_k = \{(i, k+i \bmod n) \mid 0 \leq i < n\}$. Each vertex then appears in exactly three such cycles. Secondly, given any valid 3-colouring of $\tilde{C}(T_n)$, we obtain a valid 3-colouring of T_n by assigning the colour of c_i^v to the vertex v. In particular, this implies that if we consider the colours of c_i^v in an n-cycle, then they must strictly alternate among the three colours. Thirdly, given a gadget C^v in $\tilde{C}(T_n)$, we pair off the gates of C^v into three pairs, the horizontal, vertical and diagonal, according to which n-cycle they appear in. Given a 3-colouring of $\tilde{C}(T_n)$, if the two a_i^v in one such pair are of different colours, we say that C^v makes a switch in the corresponding n-cycle. By property (3) of the gadgets C^v, it follows that each one of them makes an even number of switches. Finally, we note that for a valid 3-colouring of $\tilde{C}(T_n)$, each n-cycle must contain an even number of switches, *except* the unique n-cycle containing the twisted edge, which must contain an odd number of switches. It can be easily verified that these last two requirements lead to a contradiction.

Finally, we give an explicit construction of the gadget C_d having the properties (1)–(3) listed above. Note that, since the graph T_n is regular of degree 6, it would suffice to give a construction of C_6. However, we present a general purpose construction.

The graph C_d has $16d$ nodes altogether. It contains a spine of $3d$ nodes, $s_0, \ldots, s_{3d-1}$ with edges (s_i, s_{i+1}) and (s_i, s_{i+2}) for all i. Here, and in the rest of this section, addition in the subscripts is understood to be modulo $3d$. That is, the spine consists of a cycle of length $3d$, along with all its chords of length two. Thus, in any 3-colouring of C_d, the nodes s_i and s_{i+3} must be assigned the same colour, for all i, i.e., the colours along the spine strictly alternate among all three colours.

Next, attached to each s_i, there are two additional vertices l_i and r_i, which are also attached to each other. Morover, if $i \equiv 1 \pmod 3$ or $i \equiv 2 \pmod 3$, then l_i (resp. r_i) is connected by an edge to l_{i+1} (resp. r_{i+1}). A part of the spine is depicted in Figure 2. For clarity, the chords of length two along the spine are omitted. It can be verified from Figure 2 that given a 3-colouring of the spine, the colouring of the l_i and r_i is determined up to automorphism.

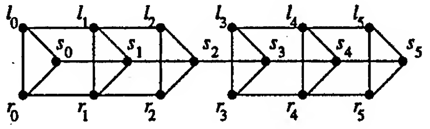


Fig. 2. A portion of the spine of C_d.

Finally, in the gap between s_{3i+2} and $s_{3(i+1)}$, we place the ith gate of C_d, by attaching the pair (a_i, b_i) to the pairs (l_{3i+2}, r_{3i+2}) and $(l_{3(i+1)}, r_{3(i+1)})$ by means of the gadget X_3 of Example 2. To ensure uniqueness of 3-colouring, we also connect a_i and b_i by edges to s_{3i+2}. An example is depicted in Figure 3, where, for clarity the edges that are part of X_3 are indicated by dashed lines. It can be verified from Figure 3 that, if the colouring of $l_{3i+2}, r_{3i+2}, l_{3(i+1)}$ and $r_{3(i+1)}$ is fixed, this also fixes the colouring of a_i and b_i.

If we now consider an automorphism that exchanges a_i and b_i, it must also exchange exactly one of the pairs (l_{3i+2}, r_{3i+2}) or $(l_{3(i+1)}, r_{3(i+1)})$. Assuming, without loss of generality, that it is the latter, the automorphism must also exchange (l_{3i+4}, r_{3i+4}) and (l_{3i+5}, r_{3i+5}) which takes us to the next gate, where we can choose to either exchange a_{i+1} and b_{i+1} or to continue to exchange nodes l and r further along the spine. In any case, we must exchange a_j and b_j for some $j \neq i$, which gives us the required properties of the gadget C_d.

Thus, having established that there exists gadgets C_d with the properties (1)–(3), the main theorem of this section follows as a consequence of Lemmas 24, 26 and 28.

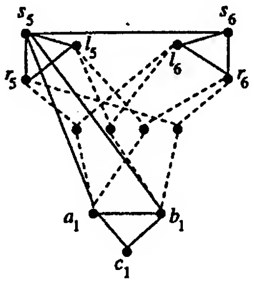


Fig. 3. A gate of C_d

Theorem 29. 3-colourability is not expressible in $L^{\omega}_{\infty\omega}$.

Remark. The proof of Theorem 29 given above can be adapted to show that 3-colourability is not even definable in the extension of $L^{\omega}_{\infty\omega}$ with counting quantifiers. To see this, note that if we add two constants c and d to our signature and interpret them in a triangular mesh T_n by two adjacent vertices in the same row, then there is an FP formula that defines a linear order in T_n. It follows that if c and d are interpreted by vertices in adjacent gadgets in $C(T_n)$ and $\tilde{C}(T_n)$, then the sizes of the $\equiv^k$ equivalence classes in these structures are bounded. Moreover, we can now even remove the constants from our signature simply by distinguishing the two gadgets in some identifiable way, say an extra vertex that does not interfere with the relevant properties of the gadget. However, when the sizes of the $\equiv^k$ equivalence classes are bounded, then counting quantifiers do not add to the expressive power of $L^{\omega}_{\infty\omega}$ (for details see [5]).

4.2 NFA Inequivalence

In this section we examine examples of natural NP-complete problems that *are* expressible in the logic $\Sigma^{1,\omega}_1$. The examples are special cases of the problem of NFA inequivalence, that is the problem of deciding, given two non-deterministic finite automata, whether or not they accept distinct languages. This problem is PSPACE-complete, and was shown by Abiteboul *et al.* [1] to be expressible in PFP. Two restrictions of this problem that are known the be NP-complete are the restriction to a finite language and the restriction to a unary alphabet (see [12]). Both of these restrictions are definable in $\Sigma^{1,\omega}_1$. We examine the second one in some detail.

The problem of determining whether two NFAs over a unary alphabet are inequivalent, which we denote UNI (for unary NFA inequivalence), can also be formulated as a problem on graphs, as follows. Given

$$N = (V, A, s_0, t_0, s_1, t_1),$$

where (V, A) is a directed graph and $s_0, s_1, t_0, t_1 \in V$ are distinguished vertices, are the two sets

$$P_0 = \{p \in \omega \mid \text{ there is a path of length } p \text{ from } s_0 \text{ to } t_0\}$$
$$P_1 = \{p \in \omega \mid \text{ there is a path of length } p \text{ from } s_1 \text{ to } t_1\}$$

distinct?

To see that this problem is definable in $\Sigma_1^{1,\omega}$, we first observe that if there is a $p \in \omega$ that distinguishes the two sets P_0 and P_1 in a structure N, then there is such a $p < 2^t$, where t is the number of distinct $\equiv^3$ equivalence classes in N. This is because, for every p, there is a formula $\psi^p(x, y)$ of L^3 which asserts that there is a path of length p from x to y. Thus, if we consider the set E^p of pairs (x, y) such that there is a path of length p from s_0 to x if, and only if, there is a path of length p from s_1 to y, then this set is $\equiv^3$-closed. Moreover, it can also be easily verified that E^{p+1} is definable from E^p. Since there are only 2^t distinct $\equiv^3$-closed sets, it follows that for $p \geq 2^t$ the sequence of sets must repeat itself. Thus, if the pair (t_0, t_1) appears in the set E^p for all $p < 2^t$, then it appears in all $E^p, p \in \omega$, which means that the sets P_0 and P_1 are identical. However, if there is a $p < 2^t$ such that $(t_0, t_1) \notin E^p$, then this p witnesses that P_0 and P_1 are distinct.

Next, we note that any number $p < 2^t$ can be represented by a pair of relations $(\leq^3, R)$, where $\leq^3$ is the ordering of $\equiv^3$-equivalence classes given by Theorem 8, and R is a $\equiv^3$-closed relation. This is done by interpreting the pair $(\leq^3, R)$ as a binary string of length t, with a 1 for each $\equiv^3$-equivalence class that is in R and a 0 for each class that is not in R. Finally, we note that, given the pair $(\leq^3, R)$, we can write a sentence $\varphi(\leq^3, R)$ of FP which asserts that the number represented by $(\leq^3, R)$ distinguishes the sets P_0 and P_1. We will not write down φ explicitly, noting only that we can write an inductive definition of a 5-ary relation S such that $S(x_1, x_2, \bar{y})$ holds if and only if, whenever $\bar{y}$ is in the ith $\equiv^3$-equivalence class, there is a path of length p_i from x_1 to x_2, where p_i is the number represented by the first i bits of $(\leq^3, R)$.

Now, it is clear that the sentence $\exists^6 O \exists^3 R \varphi(O, R)$ expresses the problem UNI. Moreover, it follows from Corollary 16, that this is equivalent to a sentence of $\Sigma_1^{1,\omega}$. This enables us to establish the following theorem.

Theorem 30. PTIME = NP *if, and only if,* UNI $\in$ FP.

Proof. In one direction, if UNI is definable in FP, it is solvable in polynomial time. Since the problem is NP-complete, this means that PTIME = NP.

In the other direction, if PTIME = NP, then by Theorem 17, $\Sigma_1^{1,\omega} = \text{FP}$. But, since UNI $\in \Sigma_1^{1,\omega}$ it follows that UNI $\in$ FP.

5 Conclusion

A great deal of research in finite model theory has been inspired by the discovery of the close connection between logical expressibility and computational complexity. This discovery raises the possibility of applying model-theoretic methods to attack outstanding open problems in complexity theory. Unfortunately, most of the results and methods that have been developed in the study of infinite models do not apply when only finite models are considered. To a large extent, the classical subject of model theory can be seen to be the study of the relation of elementary equivalence. However, this equivalence relation turns out to be of limited interest in the logical study of finite models, since it is identical with isomorphism. Recent work has shown, nonetheless, that there is an equivalence relation (or rather a countable collection of such relations), namely $\equiv^k$, which has a close connection with logical definability and which is non-trivial on finite models. Moreover, as this paper illustrates, outstanding questions in complexity theory can be reformulated in a context where this equivalence relation corresponds with the notion of definability. This further underlines the need to study the model theory of finite variable logics.

Acknowledgements: The research reported here has greatly benefited from discussions with Lauri Hella and Phokion Kolaitis. The notation introduced in Definition 9 was suggested by Scott Weinstein. I am also grateful to Lauri Hella and Victor Vianu for a careful reading of a draft of this paper.

References

1. S. Abiteboul, Moshe Y. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. In *Proc. 7th IEEE Symp. on Structure in Complexity Theory*, 1992.
2. S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.
3. S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proceedings of the 23rd ACM Symposium on the Theory of Computing*, 1991.
4. J. Barwise. On Moschovakis closure ordinals. *Journal of Symbolic Logic*, 42:292–296, 1977.
5. J-y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
6. A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.
7. E. Dahlhaus. Reduction to NP–complete problems by interpretation. In *LNCS 171*, pages 357–365. Springer-Verlag, 1984.
8. A. Dawar. *Feasible Computation through Model Theory.* PhD thesis, University of Pennsylvania, 1993.
9. A. Dawar and E.Grädel. Generalized quantifiers and 0-1 laws. Manuscript, 1994.
10. A. Dawar, S. Lindell, and S. Weinstein. Infinitary logic and inductive definability over finite structures. Technical Report MS-CIS-91-97, University of Pennsylvania, 1991. Revised version to appear in *Information and Computation.*
11. R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol 7*, pages 43–73, 1974.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

13. Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.
14. N. Immerman. Upper and lower bounds for first-order expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.
15. N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.
16. N. Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Computational Complexity Theory, Proc. of AMS Symposia in Appl. Math.*, volume 38, pages 75–91, 1989.
17. Ph. G. Kolaitis and M. Y. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, pages 46–57, 1992.
18. Ph. G. Kolaitis and M. Y. Vardi. Infinitary logics and 0-1 laws. *Information and Computation*, 98(2):258–294, 1992.
19. D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95–108, 1990.
20. L. Lovász and P. Gács. Some remarks on generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 23:27–144, 1977.
21. Y. N. Moschovakis. *Elementary Induction on Abstract Structures.* North Holland, 1974.
22. A. Rubin. *Free Algebras in Von Neumann-Bernays-Godel Set Theory and Positive Elementary Inductions in Reasonable Structures.* PhD thesis, California Institute of Technology, 1975.
23. L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.
24. M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

Comparing the Power of Monadic NP Games

Preliminary Version

Ronald Fagin

IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099
email: fagin@almaden.ibm.com

Abstract: It is well-known that the complexity class NP coincides with the class of problems expressible in existential second-order logic (Σ_1^1). *Monadic NP* is the class of problems expressible in monadic Σ_1^1, i.e., Σ_1^1 with the restriction that the second-order quantifiers range only over sets (as opposed to ranging over, say, binary relations). The author introduced a type of Ehrenfeucht-Fraïssé game, called the monadic NP game, to prove that connectivity is not in monadic NP. Later, Ajtai and the author introduced another type of monadic NP game (the "Ajtai-Fagin monadic NP game") to prove that directed reachability is not in monadic NP. Both games have two players (the spoiler and the duplicator), and involve coloring steps (where the players color nodes of the graphs) and selection steps (where the players select nodes of the graphs, round by round). It is known that the original game and the Ajtai-Fagin game are equivalent, in the sense that both characterize monadic NP. Thus, the duplicator has a winning strategy in the original game for every choice of parameters (number of colors and number of rounds) if and only if the duplicator has a winning strategy in the Ajtai-Fagin game for every choice of parameters. In this paper, we investigate the relationship between these games at a finer level. We show that in one sense, even at a finer level, Ajtai-Fagin monadic NP games are no stronger than the original monadic NP games. Specifically, we show that the families of graphs used in the Ajtai-Fagin game to prove that a problem is not in monadic NP can in principle be used in the original game to prove the same result (where for a given choice of parameters, bigger graphs of the same type are used for the original game than for the Ajtai-Fagin game). This answers an open question of Ajtai and the author. We also show that in another sense, Ajtai-Fagin games are stronger, in that there are situations where the spoiler requires more resources (colors) to win the Ajtai-Fagin game than the original game, when the choices of graphs are fixed. Our analysis gives a nonelementary upper bound, which we conjecture to be optimal, on the number of extra colors that are required for the spoiler to win the Ajtai-Fagin game than the original game.

1 Introduction

The *computational complexity* of a problem is the amount of resources, such as time or space, required by a machine that solves the problem. The *descriptive complexity* of a problem is the complexity of describing the problem in some logical formalism [Imm89]. There is an intimate connection between the descriptive complexity and the computational complexity. In particular [Fag74], the complexity class NP coincides with the class of properties of finite structures expressible in existential second-order logic, otherwise known as Σ_1^1. A consequence of this result is that NP=co-NP if and only if existential and universal second-order logic have the same expressive power over finite structures, i.e., if and only if $\Sigma_1^1 = \Pi_1^1$.

One way of attacking these difficult questions is to restrict the classes under consideration. Instead of considering Σ_1^1 (=NP) and Π_1^1 (=co-NP) in their full generality, we could consider the monadic restriction of these classes, i.e., the restriction obtained by allowing second-order quantification only over sets (as opposed to quantification over, say, binary relations). Following Fagin, Stockmeyer, and Vardi [FSV93], we refer to the restricted classes as *monadic NP* (resp., *monadic co-NP*). It should be noted that, in spite of its severely restricted syntax, monadic NP does contain NP-complete problems, such as 3-colorability and satisfiability. The hope is that the restriction to the monadic classes will yield more tractable questions and will serve as a training ground for attacking the problems in their full generality.

As a first step in this program, the author [Fag75] separated monadic NP from monadic co-NP. Specifically, it was shown that connectivity of finite graphs is not in monadic NP, although it is easy to see that it is in monadic co-NP. The proof that connectivity is not in monadic NP makes use of a certain type of Ehrenfeucht-Fraïssé game on graphs played between two players, called the spoiler and the duplicator. The game involves coloring steps (where the players color nodes of the graphs) and selection steps (where the players select nodes of the graphs, round by round). We call this game the (original) monadic NP game. In this game, the duplicator selects two graphs G_0 and G_1, where G_0 is connected and G_1 is not. The spoiler then colors G_0, and the duplicator colors G_1. They then play a first-order Ehrenfeucht-Fraïssé game on these colored graphs, where, as usual, the spoiler tries to expose differences in the graphs, and the duplicator tries to cover up these differences. A necessary and sufficient condition for proving that connectivity is not in monadic NP is to show that for each choice of parameters (number of colors and number of first-order rounds), there are graphs G_0 and G_1 on which the duplicator has a winning strategy. By showing that, indeed, the duplicator has a winning strategy, the author showed that connectivity is not in monadic NP.

Later, Ajtai and the author [AF90] continued this program by showing that (s,t)-connectivity of directed graphs (otherwise known as *directed reachability*) is not in monadic NP. They made use of a modified game, which is now often referred to as the *Ajtai-Fagin monadic NP game*. Here the duplicator selects a graph G_0 that is (s,t)-connected, and the spoiler colors G_0. Then the duplicator selects and colors a graph G_1 that is not (s,t)-connected. The game again concludes with a first-order game. The difference between the Ajtai-Fagin game and the original game is that in the Ajtai-Fagin game, the spoiler must commit himself to a coloring of G_0 before seeing G_1. Putting it another way, the duplicator can wait to decide on his choice of G_1 until he sees how the spoiler colors G_0. Because the change in rules between the original game and the Ajtai-Fagin game favors the duplicator, on the face of it the Ajtai-Fagin game is "easier for the duplicator to win", which makes it easier to prove that the duplicator has a winning strategy. In fact, Ajtai and the author introduced their variation on the original game because they did not see how to prove that the duplicator has a winning strategy in the original game. However, they were able to prove that the duplicator has a winning strategy in their variation of the game. Since the duplicator has a winning strategy, directed reachability is not in monadic NP.

There is some mystery about the relationship between the Ajtai-Fagin game and the original game. On the one hand, the two games are equivalent, in the sense that in both cases, the existence of a winning strategy for the spoiler is a necessary and sufficient condition for a class to be in monadic NP. Thus, in both cases, showing that a problem is not in monadic NP corresponds precisely to showing that the duplicator has a winning strategy. On the other hand, as we noted, the Ajtai-Fagin game seems intuitively to be easier for the duplicator to win. Because of the fundamental role of Ehrenfeucht-Fraïssé games as tools in descriptive complexity, it is important to understand better the difference in power of the Ajtai-Fagin game and the original game. In this paper, we explore this difference.

In both games, a class $\mathcal{S}$ is given (such as the class of connected graphs, or the class of (s,t)-connected graphs). Then various graphs are selected and colored by the players, and a first-order game is played on these colored graphs. The equivalence of the games corresponds to the fact that for each class $\mathcal{S}$, the duplicator has a winning strategy in the original game for every choice of parameters (number of colors and number of rounds) if and only if the duplicator has a winning strategy in the Ajtai-Fagin game for every choice of parameters.

In this paper, we investigate the relationship between the original game and the Ajtai-Fagin game at a finer level. We show that in one sense, even at a finer level, Ajtai-Fagin monadic NP games are no stronger than the original monadic NP games. This sense corresponds to fact that in a game-theoretic proof that a class is not in monadic NP, the same families of graphs can be used in the original game as in the Ajtai-Fagin game. We also show that in another sense, Ajtai-Fagin games are stronger, in that there are situations where the spoiler requires more resources (colors) to win the Ajtai-Fagin game than the original game, when the choices of graphs are fixed. We now explain the details a little more.

In a game-theoretic proof that a specific class of graphs is not in monadic NP, the duplicator inevitably restricts himself to selecting graphs only of a certain type. For example, in the case of connectivity [Fag75], the graph G_0 is a cycle, and G_1 is a disjoint union of two cycles. In the case of directed reachability

[AF90], the graph G_0 is a path from s to t along with certain backedges, and G_1 is the result of deleting one forward edge from G_0. We show that the family of graphs used in the Ajtai-Fagin game to prove that a problem is not in monadic NP can in principle be used in the original game to prove the same result (where for a given choice of parameters, bigger graphs of the same type are used for the original game than for the Ajtai-Fagin game). For example, in the case of directed reachability, we show that for every choice of parameters, the duplicator has a winning strategy in the (original) monadic NP game where the graph G_0 is a path from s to t along with certain backedges, and G_1 is the result of deleting one forward edge from G_0. This answers an open question of Ajtai and the author [AF90].

How do we obtain this result? First, we generalize the framework of the games. Rather than saying that the duplicator selects G_0 from a class $\mathcal{S}$, and selects G_1 from the complement $\overline{\mathcal{S}}$, we instead consider a more general game, where the duplicator selects G_0 from a class $\mathcal{G}_0$, and G_1 from a class $\mathcal{G}_1$. There are once again two versions, one corresponding to the original game, and one to the Ajtai-Fagin game. In the first version of this new game, the duplicator selects G_1 before the spoiler has colored G_0; in the Ajtai-Fagin version, the duplicator selects G_1 after the spoiler has colored G_0. We show that for each choice of the number c of colors and the number r of rounds, there are c' and r' such that for every choice of $\mathcal{G}_0$ and $\mathcal{G}_1$ where the duplicator has a winning strategy in the Ajtai-Fagin version of the new game with parameters c' and r', the duplicator also has a winning strategy in the first version of this game with parameters c and r (in fact, we can take $r' = r$). This result tells us that the same families of graphs can be used in the original game as in the Ajtai-Fagin game (such as to prove that a class is not in monadic NP). Intuitively, for a given choice of c, r, we use bigger graphs in the original (c, r)-game than in the Ajtai-Fagin (c, r)-game, since in the original game we use $\mathcal{G}_0, \mathcal{G}_1$ that correspond to the Ajtai-Fagin game with more colors (since $c' \geq c$).

We now consider a sense in which Ajtai-Fagin games are stronger. Here, we investigate the resources involved in the games. Specifically, we consider the number of colors required for the spoiler to win when the choices of graphs are fixed. Since the spoiler is trying to expose differences between G_0 and G_1, and the duplicator is trying to cover up these differences, it helps the spoiler for there to be more colors. We show that there are situations where the spoiler requires strictly more colors to win the Ajtai-Fagin game than the original game. Thus, in such situations, it is indeed true, in a precise sense, that it is easier for the duplicator to win the Ajtai-Fagin game than the original game.

Our analysis gives a nonelementary upper bound on the number of extra colors that are required for the spoiler to win the Ajtai-Fagin game than the original game. We conjecture that there is also a nonelementary lower bound.

This version does not contain proofs, which the interested reader can find in [Fag94].

2 Definitions and conventions

We begin with a few conventions. For convenience, we shall usually discuss only *graphs* (usually *directed* graphs, sometimes with distinguished points s and t), but everything we say can be generalized to arbitrary structures. We are also interested in "colored graphs", where each vertex has some color. (When there are 2^k possible colors for some k, it is often convenient to think of the coloring of a point as a description of which of k possible unary relations the point is a member of.) We assume throughout this paper that we are restricting our attention to *finite* graphs (and so are doing finite-model theory), although all of the results hold also without this assumption.

A Σ_1^1 *sentence* is a sentence of the form $\exists A_1 ... \exists A_k \psi$, where ψ is first-order and where the A_i's are relation symbols. As an example, we now construct a Σ_1^1 sentence that says that a graph (with edge relation denoted by P) is 3-colorable. In this sentence, the three colors are represented by the monadic relation symbols A_1, A_2, and A_3. Let ψ_1 say "Each point has exactly one color". Thus, ψ_1 is

$$\forall x((A_1x \wedge \neg A_2x \wedge \neg A_3x) \vee (\neg A_1x \wedge A_2x \wedge \neg A_3x) \vee (\neg A_1x \wedge \neg A_2x \wedge A_3x)).$$

Let ψ_2 say "No two points with the same color are connected by an edge". Thus, ψ_2 is

$$\forall x \forall y\, ((A_1x \wedge A_1y \Rightarrow \neg Pxy) \wedge (A_2x \wedge A_2y \Rightarrow \neg Pxy) \wedge (A_3x \wedge A_3y \Rightarrow \neg Pxy))\,.$$

The Σ_1^1 sentence $\exists A_1 \exists A_2 \exists A_3(\psi_1 \wedge \psi_2)$ then says "The graph is 3-colorable."

A Σ_1^1 sentence $\exists A_1 ... \exists A_k \psi$, where ψ is first-order, is said to be *monadic* if each of the A_i's is unary, that is, the existential second-order quantifiers quantify only over sets. A class $\mathcal{S}$ of graphs is said to be (*monadic*) Σ_1^1 if it is the class of all graphs that obey some fixed (monadic) Σ_1^1 sentence. One reason that Σ_1^1 classes are of great interest is the result [Fag74] that the collection of Σ_1^1 classes coincides with the complexity class NP. For this reason, as we noted earlier, we follow Fagin, Stockmeyer, and Vardi [FSV93] by referring to the collection of monadic Σ_1^1 classes as *monadic NP.* We often refer to a class of graphs by a defining property, for example, 3-colorability. As we saw above, 3-colorability is in monadic NP.

3 Ehrenfeucht-Fraïssé Games

Among the few tools of model theory that "survive" when we restrict our attention to finite structures are Ehrenfeucht-Fraïssé games [Ehr61, Fra54]. We begin with an informal definition of an *r-round first-order Ehrenfeucht-Fraïssé game* (where r is a positive integer), which we shall call an *r-game* for short. It is straightforward to give a formal definition, but we shall not do so. As we mentioned earlier, for ease in description, we shall restrict our attention to colored graphs. There are two *players*, called *the spoiler* and *the duplicator*, and two colored graphs, G_0 and G_1. In the first round, the spoiler selects a point in one of the two colored graphs, and the duplicator selects a point in the other colored graph. Let a_1 be the point selected in G_0, and let b_1 be the point selected in G_1. Then the second round begins, and again, the spoiler selects a point in one of the two colored graphs, and the duplicator selects a point in the other colored graph. Let a_2 be the point selected in G_0, and let b_2 be the point selected in G_1. This continues for r rounds. The duplicator wins if the colored subgraph of G_0 induced by $\langle a_1, \ldots, a_r \rangle$ is isomorphic to the colored subgraph of G_1 induced by $\langle b_1, \ldots, b_r \rangle$, under the function that maps a_i onto b_i for $1 \leq i \leq r$. That is, for the duplicator to win, (a) $a_i = a_j$ iff $b_i = b_j$, for each i, j; (b) $\langle a_i, a_j \rangle$ is an edge in G_0 iff $\langle b_i, b_j \rangle$ is an edge in G_1, for each i, j; and (c) a_i has the same color as b_i, for each i. Otherwise, the spoiler wins. We say that the spoiler or the duplicator *has a winning strategy* if he can guarantee that he will win, no matter how the other player plays. Since the game is finite, and there are no ties, the spoiler has a winning strategy iff the duplicator does not. If the duplicator has a winning strategy, then we write $G_0 \sim_r G_1$. In this case, intuitively, G_0 and G_1 are indistinguishable by an r-game.

We now discuss a more complicated game, which is a c-color, r-round, monadic NP game, and which we shall call a *(c, r)-game* for short. This game was introduced in [Fag75] to prove that connectivity is not in monadic NP. We start with two graphs G_0 and G_1 (in this case, not colored). Let C be a set of c distinct colors. The spoiler first colors each of the points of G_0, using the colors in C, and then the duplicator colors each of the points of G_1, using the colors in C. Note that there is an asymmetry in the two graphs in the rules of the game, in that the spoiler must color the points of G_0, not G_1. The game then concludes with an r-game. The duplicator now wins if the colored subgraph of G_0 induced by $\langle a_1, \ldots, a_r \rangle$ is isomorphic to the colored subgraph of G_1 induced by $\langle b_1, \ldots, b_r \rangle$, under the function that maps a_i onto b_i for $1 \leq i \leq r$.

Theorem 1. *[Fag75] Let $\mathcal{S}$ be a class of graphs. $\mathcal{S}$ is in monadic NP iff there are c, r such that whenever $G_0 \in \mathcal{S}$ and $G_1 \in \overline{\mathcal{S}}$, then the spoiler has a winning strategy in the (c, r)-game over G_0, G_1.*

In [Fag75] it is shown that given c and r, there is a graph G_0 that is a cycle, and a graph G_1 that is the disjoint union of two cycles, such that the duplicator has a winning strategy in the (c, r)-game over G_0, G_1. Since G_0 is connected and G_1 is not, it follows from Theorem 1 that connectivity is not in monadic NP.

In addition to considering games over pairs G_0, G_1 of graphs, Ajtai and the author [AF90] found it convenient, for reasons we shall see shortly, to consider games over a class $\mathcal{S}$. The rules of a (c, r)-game over $\mathcal{S}$ are as follows.

1. The duplicator selects a member of $\mathcal{S}$ to be G_0.
2. The duplicator selects a member of $\overline{\mathcal{S}}$ to be G_1.
3. The spoiler colors G_0 with the c colors.
4. The duplicator colors G_1 with the c colors.
5. The spoiler and duplicator play an r-game on the colored G_0, G_1.

The next theorem follows easily from Theorem 1.

Theorem 2. *Let $\mathcal{S}$ be a class of graphs. $\mathcal{S}$ is in monadic NP iff there are c, r such that the spoiler has a winning strategy in the (c, r)-game over $\mathcal{S}$.*

We now explain why Ajtai and the author allowed G_0 and G_1 to be selected by the duplicator, rather than inputs to the game. A directed graph with distinguished points s, t is said to be (s, t)*-connected* if there is a directed path in the graph from s to t. Ajtai and the author wished to prove that directed (s, t)-connectivity (also known as directed reachability) is not in monadic NP, but they did not see how to prove this by using (c, r)-games. By considering the choice of G_0 and G_1 to be moves of the duplicator, rather than inputs to the game, they were able to define a variation of (c, r)-games, in which the choice of G_1 by the duplicator is delayed until after the spoiler has colored G_0. They successfully used the new game to prove the desired result (that directed reachability is not in monadic NP). Their new game, which is usually called the *Ajtai-Fagin* (c, r)*-game*, is, on the face of it, easier for the duplicator to win. The rules of the new game are obtained from the rules of the (c, r)-game by reversing the order of the second and third moves. Thus, the rules of the Ajtai-Fagin (c, r)-game are as follows.

1. The duplicator selects a member of $\mathcal{S}$ to be G_0.
2. The spoiler colors G_0 with the c colors.
3. The duplicator selects a member of $\overline{\mathcal{S}}$ to be G_1.
4. The duplicator colors G_1 with the c colors.
5. The spoiler and duplicator play an r-game on the colored G_0, G_1.

The winner is decided as before. Thus, in the Ajtai-Fagin (c, r)-game, the spoiler must commit himself to a coloring of G_0 with the c colors before knowing what G_1 is. In order to contrast it with the Ajtai-Fagin (c, r)-game, we may sometimes refer to the (c, r)-game as the *original* (c, r)-game (or the *original* monadic NP game). In spite of the fact that it seems to be harder for the spoiler to win the Ajtai-Fagin (c, r)-game than the original (c, r)-game, we have the following analogue to Theorem 2.

Theorem 3. *[AF90] Let $\mathcal{S}$ be a class of graphs. $\mathcal{S}$ is in monadic NP iff there are c, r such that the spoiler has a winning strategy in the Ajtai-Fagin (c, r)-game over $\mathcal{S}$.*

The next theorem is an immediate consequence of Theorems 2 and 3:

Theorem 4. *Let $\mathcal{S}$ be a class of graphs. The following are equivalent.*

1. *For every c, r, the duplicator has a winning strategy in the original (c, r)-game over $\mathcal{S}$.*
2. *For every c', r', the duplicator has a winning strategy in the Ajtai-Fagin (c', r')-game over $\mathcal{S}$.*

Theorem 4 gives a precise sense in which the original monadic NP game and the Ajtai-Fagin monadic NP game are equivalent. Later, we shall see stronger versions of this equivalence.

4 Inseparability

In this section, we introduce a notion of *inseparability.* In this section and the next section, we use inseparability to give stronger versions of the equivalence between the original monadic NP game and the Ajtai-Fagin monadic NP game. This notion of inseparability allows us to make sense of the notion of "the graphs used in a game", so that we can consider statements such as "the same family of graphs used in the Ajtai-Fagin monadic NP game can be used in the original monadic NP game to prove that a problem is not in monadic NP."

We define a variation of the original (c,r)-game over a class $\mathcal{S}$, by replacing $\mathcal{S}$ and $\overline{\mathcal{S}}$ by arbitrary classes $\mathcal{G}_0$ and $\mathcal{G}_1$ of graphs. Thus, let $\mathcal{G}_0$ and $\mathcal{G}_1$ be classes of graphs, and let c, r be positive integers. We define the *original* (c,r)*-game over* $\mathcal{G}_0, \mathcal{G}_1$ to have the following rules.

1. The duplicator selects a member of $\mathcal{G}_0$ to be G_0.
2. The duplicator selects a member of $\mathcal{G}_1$ to be G_1.
3. The spoiler colors G_0 with the c colors.
4. The duplicator colors G_1 with the c colors.
5. The spoiler and duplicator play an r-game.

The winner is decided as before.

We say that $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)*-inseparable*$_1$ if the duplicator has a winning strategy in the original (c,r)-game over $\mathcal{G}_0, \mathcal{G}_1$. In particular, if the duplicator has a winning strategy in the original (c,r)-game over $\mathcal{S}$, then $\mathcal{S}, \overline{\mathcal{S}}$ are (c,r)-inseparable$_1$. If it is the spoiler, rather than the duplicator, who has a winning strategy, then we say that $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)*-separable*$_1$.

Similarly to before, we define the *Ajtai-Fagin* (c,r)*-game over* $\mathcal{G}_0, \mathcal{G}_1$ by exchanging the order of the second and third moves in the original game. As before, the difference between the original (c,r)-game and Ajtai-Fagin (c,r)-game is that in the Ajtai-Fagin game, the spoiler must commit himself to a coloring of G_0 before knowing which graph the duplicator selects as G_1.

We have the following strengthened version of Theorem 4.

Theorem 5. *Let $\mathcal{G}_0, \mathcal{G}_1$ be classes of graphs. The following are equivalent.*

1. *For every c, r, the duplicator has a winning strategy in the original (c,r)-game over $\mathcal{G}_0, \mathcal{G}_1$.*
2. *For every c', r', the duplicator has a winning strategy in the Ajtai-Fagin (c', r')-game over $\mathcal{G}_0, \mathcal{G}_1$.*

We say that $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)*-inseparable*$_2$ (resp., (c,r)*-separable*$_2$) if the duplicator (resp., spoiler) has a winning strategy in the Ajtai-Fagin (c,r)-game over $\mathcal{G}_0, \mathcal{G}_1$.

The next proposition follows immediately from the definitions. It says, intuitively, that if the duplicator has a winning strategy in the original monadic NP game, then he has a winning strategy in the Ajtai-Fagin monadic NP game, with the same choices of graphs. This is what we would expect, since intuitively, it is even easier for the duplicator to win the Ajtai-Fagin game than the original game.

Proposition 6. *Let $\mathcal{G}_0, \mathcal{G}_1$ be classes of graphs. If $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)-inseparable$_1$, then $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)-inseparable$_2$.*

As we shall see (Theorem 9), the converse is false. We are interested in comparing inseparability$_1$ and inseparability$_2$ to compare the graphs that can be used in a proof that a property is not in monadic NP using Ajtai-Fagin monadic NP games and using the original monadic NP games. An example of this reasoning appears in Section 6.

5 Ajtai-Fagin games are no stronger

The next theorem is a strengthening of Theorem 5. It is a partial converse to Proposition 6. It tells us that for each c, r, there are c', r' such that if $\mathcal{G}_0, \mathcal{G}_1$ are (c', r')-inseparable$_2$, then $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)-inseparable$_1$. In fact, we can let $r' = r$. As we shall see (Theorem 9), we cannot always let $c' = c$. Hence, the converse of Proposition 6 is false, so we must settle for a partial converse.

Theorem 7. *Let c, r be positive integers. There is c' such that for every $\mathcal{G}_0, \mathcal{G}_1$ that are (c', r)-inseparable$_2$, also $\mathcal{G}_0, \mathcal{G}_1$ are (c,r)-inseparable$_1$.*

This theorem is rather powerful, since it guarantees the existence of a winning strategy for the duplicator in the original game (with a certain choice of parameters) given only the existence of a winning strategy for the duplicator in the Ajtai-Fagin game (with another choice of parameters). Intuitively, for a given choice of c, r, we use bigger graphs in the original (c,r)-game than in the Ajtai-Fagin (c,r)-game, since in the original game we use $\mathcal{G}_0, \mathcal{G}_1$ that correspond to the Ajtai-Fagin game with more colors. Note that the choice of c' is uniform, over all possible choices of $\mathcal{G}_0, \mathcal{G}_1$. As we shall see by example in the next section, Theorem 7 tells us that the same families of graphs can be used in the original game as in the Ajtai-Fagin game (such as to prove that a class is not in monadic NP).

6 Directed reachability

As we noted earlier, Ajtai and the author introduced their variation of monadic NP games in order to prove that directed reachability is not in monadic NP. In this section, we discuss this approach, and in particular discuss various senses in which the original monadic NP game is adequate and is not adequate to obtain this result.

Let c and r be given. Ajtai and the author constructed (by probabilistic methods) a finite directed graph G_0 with points s, t where there is a directed path from s to t in G_0. In fact, G_0 consists of a path from s to t (these edges in the path are called "forward edges"), along with certain backedges. Thus, G_0 is (s,t)-connected. Denote the graph that is obtained by deleting the edge e from G_0 by $G_0 - e$. In particular, if e is a forward edge, then $G_0 - e$ is not

(s,t)-connected. Ajtai and the author showed that however the spoiler colors G_0 with the c colors, there is a forward edge e of G_0 such that when $G_1 = G_0 - e$ is colored in precisely the same way, vertex for vertex, as G_0 was colored, then the duplicator has a winning strategy in the r-game played on G_0 and G_1 (where, as before, the isomorphism must also respect color). Since G_0 is (s,t)-connected and G_1 is not, it follows from Theorem 3 that directed reachability is not in monadic NP.

Note that the duplicator does not commit himself to a choice of G_1 until the spoiler has committed himself to a coloring of G_0. This is the power of Ajtai-Fagin monadic NP games.

It is interesting to see what would happen if we were to try to use these pairs G_0, G_1 in the original monadic NP game rather than the Ajtai-Fagin monadic NP game. Intuitively, in the original game, the spoiler knows what G_0 and G_1 are before he colors G_0. This would be disastrous for a duplicator whose coloring strategy is to color G_1 by simply duplicating the coloring for G_0: if the spoiler knew which edge e were deleted from G_0 to form $G_1 = G - e$, this might dramatically influence his coloring of G_0 (for example, the spoiler might color the endpoints of e with special colors). In the Ajtai-Fagin monadic NP game, the spoiler must commit himself to a coloring of G_0 before he knows which edge e is deleted. This makes it easier for the duplicator to win.

Ajtai and the author commented that they do not know how to prove their main result (that directed reachability is not in monadic NP) by using the original game. In such a proof, it would be necessary, given c, r, to show the existence of a pair $\widehat{G_0}, \widehat{G_1}$ of finite directed graphs where $\widehat{G_0}$ is (s,t)-connected, $\widehat{G_1}$ is not (s,t)-connected, and the duplicator has a winning strategy in the (c,r)-game over $\widehat{G_0}, \widehat{G_1}$. Since directed reachability is not in monadic NP (as Ajtai and the author showed), it follows from Theorem 1 that for each pair c, r, there is such a pair $\widehat{G_0}, \widehat{G_1}$. Ajtai and the author instead used Ajtai-Fagin monadic NP games, and worked with pairs G_0, G_1 where G_0 consists of a path from s to t, along with certain backedges, and where G_1 is the result of deleting some forward edge from G_0. Ajtai and the author said that it is not clear that such a pair G_0, G_1 could serve as $\widehat{G_0}, \widehat{G_1}$. The next theorem, which follows easily from Theorem 7, resolves this question by saying that such a pair G_0, G_1 could serve as $\widehat{G_0}, \widehat{G_1}$.

Theorem 8. *For every c, r, there is a graph G_0 that consists of a path from s to t, along with certain backedges, and a graph G_1 that is the result of deleting some forward edge from G_0, such that the duplicator has a winning strategy in the (c,r)-game over G_0, G_1.*

It is important to note that in spite of Theorem 8, we do *not* know a direct proof, using only the original monadic NP game, that directed reachability is not in monadic NP.

Remark. We seem to need to use Theorem 7, rather than the slightly weaker Theorem 5, to prove Theorem 8. What could we obtain by using only Theorem 5? Let $\mathcal{G}_0$ consist of all graphs that are a path from s to t, along with some backedges, and let $\mathcal{G}_1$ be the collection of all graphs that are the result of deleting an arbitrary forward edge from an arbitrary member of $\mathcal{G}_0$. We know from [AF90] that for every c', r', the duplicator has a winning strategy in the Ajtai-Fagin (c', r')-game over $\mathcal{G}_0, \mathcal{G}_1$. It follows from Theorem 5 that for every c, r, the duplicator has a winning strategy in the original (c, r)-game over $\mathcal{G}_0, \mathcal{G}_1$. Therefore, for every c, r, there are $G_0 \in \mathcal{G}_0$ and $G_1 \in \mathcal{G}_1$ such that the duplicator has a winning strategy in the (c, r)-game over G_0, G_1. But this result is not as strong as Theorem 8, since G_1 might be the result of deleting some forward edge from some member of $\mathcal{G}_0$ other than G_0.

7 Ajtai-Fagin games are stronger

The next theorem says that inseparability$_1$ and inseparability$_2$ are different. Thus, the converse of Proposition 6 is false. This tells us that there are situations where the spoiler requires strictly more colors to win the Ajtai-Fagin game than the original game.

Theorem 9. *There are classes $\mathcal{G}_0$ and $\mathcal{G}_1$ of graphs, and constants c, r, such that $\mathcal{G}_0$ and $\mathcal{G}_1$ are (c, r)-separable$_1$, but $\mathcal{G}_0$ and $\mathcal{G}_1$ are (c, r)-inseparable$_2$.*

The proof of Theorem 9 shows that there are classes $\mathcal{G}_0, \mathcal{G}_1$ of graphs that are $(2,2)$-separable$_1$ but $(2,2)$-inseparable$_2$. It turns out that $\mathcal{G}_0, \mathcal{G}_1$ are $(3,2)$-separable$_2$. So in this example, 3 colors are required for the spoiler to have a winning strategy in the Ajtai-Fagin game, but only 2 colors to have a winning strategy in the original game. Hence, in this case, the Ajtai-Fagin game is harder for the spoiler to win (and therefore easier for the duplicator to win) than the original game.

8 How many more colors are required?

The contrapositive of Theorem 7 says that for each choice of the number c of colors and the number r of rounds, there is c' such that whenever $\mathcal{G}_0, \mathcal{G}_1$ are (c, r)-separable$_1$, then $\mathcal{G}_0, \mathcal{G}_1$ are (c', r)-separable$_2$. Let us denote the minimal such value of c' by $F(c, r)$. Intuitively, when the spoiler can win the original game with c colors and r rounds, then $c' = F(c, r)$ is the number of colors the spoiler needs to win the Ajtai-Fagin game. We saw in Theorem 9 that there are c, r where $F(c, r) > c$. This says that the spoiler requires strictly more colors to win the Ajtai-Fagin game than the original game. How many more colors are required? That is, how much bigger does $F(c, r)$ need to be than c? We now give an upper bound on $F(c, r)$. Define f by letting $f(0) = 2r^2 + r \log_2 c$, and $f(m+1) = 2^{f(m)}$ for each m.

Theorem 10. $F(c, r) \leq cf(r + 2)$.

Note that the upper bound $cf(r+2)$ in Theorem 10 contains a tower of $r+2$ exponents, where the top exponent is a polynomial in r and $\log_2 c$. This represents a nonelementary growth rate. We conjecture that there is a corresponding nonelementary lower bound.

Acknowledgments: The author is grateful to Phokion Kolaitis and Moshe Vardi for helpful discussions.

References

[AF90] M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected finite graphs. *Journal of Symbolic Logic*, 55(1):113–150, March 1990.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math.*, 49:129–141, 1961.

[Fag74] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[Fag75] R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[Fag94] R. Fagin. Comparing the power of monadic NP games. Research Report RJ 9908, IBM, 1994. Version 1.1.

[Fra54] R. Fraïssé. Sur quelques classifications des systèmes de relations. *Publ. Sci. Univ. Alger. Sér. A*, 1:35–182, 1954.

[FSV93] R. Fagin, L. Stockmeyer, and M. Y. Vardi. On monadic NP vs. monadic co-NP. In *Proc. 8th IEEE Conf. on Structure in Complexity Theory*, pages 19–30, 1993. To appear in *Information and Computation.*

[Imm89] N. Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Computational Complexity Theory, Proc. Symp. Applied Math., Vol. 38*, pages 75–91. American Mathematical Society, 1989.

Linear Constraint Query Languages Expressive Power and Complexity

Stéphane Grumbach[1] and Jianwen Su[2] and Christophe Tollu[3]

[1] University of Toronto and INRIA[§]
[2] University of California at Santa Barbara[¶]
[3] Université Paris-Nord, Villetaneuse[||]

Abstract. We give an AC^0 upper bound on the complexity of first-oder queries over (infinite) databases defined by restricted linear constraints. This result enables us to deduce the non-expressibility of various usual queries, such as the parity of the cardinality of a set or the connectivity of a graph in first-order logic with linear constraints.

1 Introduction

Since its inception in the early 70's, Codd's relational model of data [Cod70] has been the standard framework of much work on relational databases and query languages. The almost contemporary renewal of "finite model theory" (which dates back to the Ph.D. dissertation of Ron Fagin in 1973) has offered a logical counterpart to this development. So far, Finite Model Theory has been chiefly concerned with the study of extensions of first-order theory and has greatly contributed to a better understanding of the expressibility and the complexity of relational query languages. In short, Finite Model Theory and Codd's relational model have proven to be quite appropriate to the study and design of languages for systems manipulating finite relational data. But, since they compel all relations to be effectively represented, they are no longer adequate to new applications in databases, such as spatial (geographic) or temporal databases, which obviously require the use of infinite sets. Of course, it is unreasonable, from a mere computational point of view, to jump directly from the class of all finite structures to the class of all countable structures. For example, a straightforward extension of the relational model would require infinite representation(s) of infinite data. One must consider more subtle (and more efficient) generalizations, where the data are handled by "finite means".

[§] I.N.R.I.A., Rocquencourt BP 105, 78153 Le Chesnay, France – Stephane.Grumbach-@inria.fr – Work supported in part by Esprit Project BRA AMUSING, and an NSERC fellowship in Canada.

[¶] Dept. of Computer Science, Univ. of California, Santa Barbara, CA 93106, USA – su@cs.ucsb.edu – Work supported in part by NSF grant IRI-9117094 and NASA grant NAGW-3888. A part of work was done while visiting I.N.R.I.A.

[||] LIPN-URA 1507, Institut Galilée, Université Paris-Nord, 93430 Villetaneuse, France – Christophe.Tollu@lipn.univ-paris13.fr

Such generalizations have been the subject of various attempts in recent years; the most promising ones draw their inspiration from already established research areas either in logic (the study of recursive structures in classical model theory and effective algebra) or in computer science (the constraint programming paradigm).

Recursive structures (i.e. relational structures over a countable domain, say the set of natural numbers, where every relation is a recursive set of tuples) have been presented by Hirst and Harel [HH93] as a good alternative to finite structures. They have come up with an important trade-off between the class of structures taken as semantics and the class of admissible queries, which poses the challenging problem of exhibiting interesting classes that lie between the recursive and the highly symmetric ones.

The *constraint database model*, introduced by Kanellakis, Kuper and Revesz in their seminal paper [KKR90] and convincingly advocated in [KG94], is another powerful generalization of Codd's relational model. In this new paradigm, instead of tuples, queries act on "generalized tuples" expressed as quantifier-free first-order constraints in a decidable theory adequate to definite purposes. A generalized (or *finitely representable* in our terminology) relation is a conjunction of such constraints, interpreted in the domain of a given model of the decidable theory. Interesting (and hopefully powerful enough) constraint query languages are therefore obtained by coupling the relational calculus or some version of Datalog with the theory of dense linear orders or the theory of real closed fields.

The expressive power and the complexity of first-order logic over finitely representable databases is still far from being clearly understood. Nonetheless, a series of complexity and/or expressibility bounds have been exhibited in [KKR90, KG94, GS94, GS95]. In particular, Kanellakis and Goldin have thoroughly investigated the class of constraints expressed in $\mathcal{L} = \{=, \leqslant\}$ over a dense order and shown that every first-order query (in $\mathcal{L}$) over such constraint databases can be computed in constant parallel time (uniform AC^0) with respect to the size of the database. The latter result, combined with lower bounds on the complexity of queries like Parity and Connectivity immediately yields non-expressibility corollaries [GS94]. It seems highly probable that similar non-expressibility results still hold when the language of constraints is enriched with addition and even multiplication. In the present paper, we aim to make one step forward in this direction by considering linear constraint (expressed in $\{=, \leqslant, +\}$) instead of dense-order ones. We shall not be able to produce a similar complexity upper bound for the full case (linear first-order queries over linear constraint databases). Fortunately enough, we exhibit a restricted class of linear constraint databases to which Kanellakis and Goldin's AC^0 upper bound can be extended. The main results can be summed up as follows ($\mathbb{Z}$ is the set of integers and $\mathbb{Q}$ the set of rationals, other notions will be defined in the following sections):

Theorem 5.2 Every first-order query in $\{=, \leqslant, +\} \cup \mathbb{Q}$ over structures finitely representable in $\{=, \leqslant, +\} \cup \mathbb{Z}$ with the number of occurrences of $+$ in every constraint uniformly bounded, can be evaluated in AC^0.

The previous theorem is proved assuming a binary encoding of the integers. It does not carry over in the general case with no uniform bound on the number of occurrences of + in every constraint in the inputs. We can therefore conclude that the data complexity of first-order queries over linear constraint databases is not in AC^0 in general. Kanellakis and Goldin [KG95] suggested to study the data complexity of first-order queries over linear constraint databases in the case where integers are encoded in unary. We prove that under the latter encoding assumption, the AC^0 upper-bound holds in the general case. We think that the theorem proven here constitutes a significant improvement since linear constraints are far more expressive than the dense-order ones. As a consequence, we get the following corollary.

Theorem 6.1 Parity, graph connectivity, and region connectivity are not first-order definable with linear constraints.

Note that the first-order undefinability of parity and graph connectivity has been obtained independently by Paredaens, Van den Bussche and Van Gucht [PVV95]. The main theorem (Theorem 5.2) does not carry over in presence of multiplication. Nevertheless, we conjecture that its corollary (Theorem 6.1) holds for polynomial constraints. Proofs in this paper are made in the case of the rational numbers. The undefinability results carry over in the case of linear constraints over other domains such as the natural numbers, the integers, or the reals for instance.

The paper is structured as follows. In Section 2, we review and discuss some results aiming at initiating an elementary model theory for different classes of countable structures. Section 3 is devoted to basic definitions and examples of finitely representable databases. Section 4 exhibits an algebraic language that is a procedural equivalent of first-order logic over finitely representable databases. The algebra is used in Section 5 to prove the main theorem, from which we infer the non-expressibility results of Section 6. Throughout the paper, we assume familiarity with complexity classes defined by families of boolean circuits, especially NC (functions computable in polylogarithmic time with a polynomial amount of hardware) and AC^0 (functions computable in constant time with a polynomial amount of hardware). For more details on complexity classes, we refer to [Joh90].

2 Restricted Classes of Models

In this section, we emphasize some logical consequences of the decision to work with subclasses of countable models. In particular, we investigate conditions under which the compactness or the completeness theorem do not hold. It has been known for long that restricting oneself to finite structures ruins compactness and completeness. On the contrary, extending the semantics to all countable structures ensures compactness (a direct consequence of the Löwenheim-Skolem Theorem). In this section, we fix a purely relational signature $\sigma = \{R_1, \ldots, R_n\}$

(sometimes, one needs that at least one of the R_i's is of arity $\geqslant 2$). All structures will be of the form $\mathcal{A} = \langle A, R_1, \ldots, R_n \rangle$, with A some countable set (say a subset of natural numbers). If A is finite, one recovers the usual notion of a *finite* structure. If A and all R_i's are recursively enumerable (respectively recursive, primitive recursive), then $\mathcal{A}$ is said to be *recursively enumerable* (respectively *recursive*, *primitive recursive*). Let Str_{fin} (respectively $Str_{r.e.}$, Str_{rec}, $Str_{p.r.}$) denote the set of all finite (respectively recursively enumerable, recursive, primitive recursive) structures, and V_{fin} (respectively $V_{r.e.}$, V_{rec}, $V_{p.r.}$) denote the set of all σ-sentences true in all structures of Str_{fin} (respectively $Str_{r.e.}$, Str_{rec}, $Str_{p.r.}$). The following theorem, due to Mostowski [Mos57] and Vaught [Vau60], establishes that, for any reasonable class of "constructive structures", the completeness theorem fails:

Theorem 2.1 (Mostowski [Mos57] and Vaught [Vau60]) Let V be a set of σ-sentences. If $V_{r.e.} \subseteq V \subseteq V_{fin}$, then V is not recursively enumerable. Moreover, if $V_{r.e.} \subseteq V \subseteq V_{p.r.}$, then V is not arithmetical.

Let us now consider with more details the class Str_{rec}. For $\sigma = \{E\}$, where E is binary, it has already been the focus of some attention in the past, particularly from combinatorists. Indeed, it has been proved that switching from finite graphs to recursive ones can tremendously increase the data complexity of usual problems. For instance, the existence of a Euler path (which can be decided in polynomial time in the finite case) becomes Π^0_3-complete, thus undecidable [Bea76], while Hamiltonicity (a well-known NP-complete problem for finite graphs) becomes Σ^1_1-complete, thus even not in the arithmetical hierarchy [Har91].

More recently, Hirst and Harel [HH93] studied the recursive structures from a database point of view. Some of their results are worth mentioning. It is known that very primitive relational operators, e.g. projections, do not preserve the recursiveness of relations: if $T(x, y, z) \subset \omega^3$ is the primitive recursive relation expressing that "the y^{th} Turing machine halts on input z in x steps", then $\exists x\, T(x, y, z) \subset \omega^2$ expresses the halting problem. As a consequence, if one wants queries to be computable, even the relational calculus (i.e. first-order logic) is too expressive a language. Hirst and Harel show that, over the class of all recursive countable databases, quantifier-free first-order logic is complete with respect to the class of computable and generic (a consistency criterion expressing commutation with isomorphisms) queries. Consequently, they define a drastically restricted subclass of recursive databases, called "highly symmetric", whose behavior with respect to completeness (a version of Chandra and Harel's QL [CH80]) and BP-completeness [Ban78, Par78] (first-order logic) resemble the class of finite databases. Thus, they have come up with an important trade-off between the class of structures taken as semantics and the class of admissible queries, which poses the challenging problem of exhibiting interesting classes that lie between the recursive and the highly symmetric ones. Seemingly, the constraint database model offers a framework for the definition of such classes.

In their notes on recursive model theory [HH94], Hirst and Harel prove that the compactness theorem fails for the class of all countable recursive structures.

Their argument does not lend itself naturally to arbitrary subclasses of countable structures. J. Väänänen [Va94] suggested that the compactness theorem should fail for any subclass of countable structures containing all finite structures and no infinite countable structure elementary equivalent to a fixed (infinite) locally finite structure (a structure is locally finite if every sentence of its theory has a finite model).

3 Linear Constraint Databases

Constraint databases may be defined over various sorts of constraints, such as dense-order constraints, polynomial constraints over the reals, etc. Here we introduce a general paradigm independent of the choice of the constraints. Let $\mathcal{L}$ be a first-order language with equality and D some non empty set. We consider an $\mathcal{L}$-structure, $\mathcal{D}$, with universe D. $\mathcal{D}$ is called the *domain-structure.* Finally, let $\mathcal{T}$ be the first-order theory of $\mathcal{D}$.

Consider for instance, $\mathcal{L} = \{\leqslant, +\} \cup \mathbb{Q}$. The structure we shall be concerned with in the present paper is $\mathcal{D} = \langle \mathbb{Q}, \leqslant, +, (q)_{q \in \mathbb{Q}} \rangle$, the structure of the linearly ordered set of the rational numbers with addition and all rational constants, and $\mathcal{T}$ is the theory of dense orders without endpoints and with addition. [Another traditional example is $\mathcal{L} = \{\leqslant, +, \times, 0, 1\}$, $\mathcal{D} = \langle \mathbb{R}, \leqslant, +, \times, 0, 1 \rangle$ (the field of reals) and $\mathcal{T}$ is the theory of the ordered real closed fields.]

Let $\sigma = \{R_1, ..., R_n\}$ be a signature (or a database schema) such that $\mathcal{L} \cap \sigma = \varnothing$, where $R_1, ..., R_n$ are relation symbols. We distinguish between *logical predicates* (e.g., $=, \leqslant$) in $\mathcal{L}$ and *relations* in σ. We next introduce a restricted definition of *finitely representable structures* [GS94]. We consider expansions of $\mathcal{D}$ to σ. Intuitively, the relations in σ constitute a database *in the context* of $\mathcal{D}$.

Definition 3.1 Let $S \subseteq D^k$ be some k-ary relation. The relation S is *finitely representable in $\mathcal{L}$ over $\mathcal{D}$* (*$\mathcal{L}$-representable* for short) if there exists a quantifier free formula $\varphi(x_1, ..., x_k)$ in $\mathcal{L}$ with k distinct free variables $x_1, ..., x_k$ such that:

$$\forall a_1, ..., a_k \in D, \quad (a_1, ..., a_k) \in S \iff \mathcal{D} \models \varphi(a_1, ..., a_k)$$

Let $\mathcal{A}$ be an expansion of $\mathcal{D}$ to σ. The structure $\mathcal{A}$ is finitely representable (over $\mathcal{D}$) if for every relation symbol R in σ, $R^{\mathcal{A}}$ is $\mathcal{L}$-representable (over $\mathcal{D}$).

Kanellakis, Kuper, and Revesz [KKR90] introduced the concept of a k-ary generalized tuple, which is a constraint expressed as a conjunction of atomic formulas in $\mathcal{L}$ over k variables. A k-ary finitely representable relation (or generalized relation in [KKR90]) is then a finite set of k-ary generalized tuples. In the remainder of the paper, we focus on the language $\mathcal{L} = \{\leqslant, +\} \cup \mathbb{Q}$ and the $\mathcal{L}$-structure $\mathcal{D} = \langle \mathbb{Q}, \leqslant, +, (q)_{q \in \mathbb{Q}} \rangle$. Therefore, constraints will be composed of linear equations or inequalities of the form:

$$\sum_{i=1}^{p} a_i x_i = a_0, \qquad \sum_{i=1}^{p} a_i x_i \leqslant a_0$$

where the x_i's denote variables and the a_i's are integer constants (note that rational constants can always be avoided in linear equations and inequalities).

A *(database) instance (of σ)* is a mapping which associates with each k-ary relation symbol R in σ a quantifier-free formula in disjunctive normal form (DNF) with k distinct variables. Clearly, each instance of σ corresponds to the restriction of a finitely representable structure to σ. In practice, we assume that the databases contain the formula defining their relations. Instances will be denoted by I, J, etc.

Note that the class $\mathcal{K}$ of σ-instances is effectively enumerable if the cardinality of the language $\mathcal{L}$ is countable. Moreover, if $\mathcal{D}$ is recursive, then instances are recursive. $\mathcal{K}$ has interesting closure properties. It is closed under finite union and intersection and moreover under complementation. This differs from finite model theory (the complement of a finite model is not finite). Our main goal is to investigate the expressive power of first-order logic over the class of linear constraint databases. We consider partial recursive classes of $\mathcal{L}$-representable databases and ask whether they can be captured by a first-order sentence in $\mathcal{L}$.

In the main theorem, we restrict our attention to a family of database instances, called "k-bounded" instances. Intuitively, k-bounded linear instances are defined with equations and inequalities with bounded variable factors. We shall prove that first order queries over k-bounded instances can be evaluated in AC^0 in terms of the database size (data complexity). Following is a formal definition of k-boundedness.

Definition 3.2 Let $k \geqslant 0$ be an integer. An atomic formula is *k-bounded* if it is in $\{\leqslant, +\} \cup \mathbb{Z}$ (no rationals) and contains at most k occurrences of the (function) symbol "+". A quantifier-free formula is *k-bounded* if each atomic formula in it is k-bounded. Finally, an instance of signature σ is *k-bounded* if for each relation symbol R, the associated quantifier-free formula is k-bounded. We denote by $\mathcal{K}_k(\sigma)$, or simply $\mathcal{K}_k$, the family of all k-bounded instances over σ.

A k-bounded constraint has the following form:

$$\left(\sum_{i=1}^{p} a_i\, x_i\right)\ \Theta\ a_0$$

where Θ is a predicate, the a_i's are integers, and $\sum_{i=1}^{p} |a_i| + \overline{a}_0 \leqslant k+2$ (where $|a_i|$ denotes the absolute value of a_i, and $\overline{a}_0 = 1$ if $a_0 \neq 0$, and $\overline{a}_0 = 0$ otherwise).

Note that when $k = 0$, $\mathcal{K}_0$ is exactly the set of dense order constraints which were studied in [KKR90, KG94, GS94]. For this class of constraints, an upper bound on the complexity of the first-order queries expressed in the language $\{\leqslant\} \cup \mathbb{Q}$ is known:

Theorem 3.1 [KG94] The data complexity of first order logic in the language $\{\leqslant\} \cup \mathbb{Q}$ over the family $\mathcal{K}_0$ of dense order instances is in AC^0.

The proof of this result is based on a canonical encoding of dense order instances into finite instances. This is possible since dense order instances admit very simple geometrical decompositions in terms of atomic "cells" [Col75] of simple shapes. Note that the encoding itself is not in AC^0. A specific algebra working on finite structures is introduced in [KG94], which simulates the manipulation of dense order instances.

4 First-order Query Languages

We define $\mathrm{FO}_{\mathcal{L}}$ as first-order logic with linear constraints, i.e. over the language $\mathcal{L} = \{\leqslant, +\} \cup \mathbb{Q}$. We introduce in this section an algebra $\mathrm{ALG}_{\mathcal{L}}$ for finitely representable databases, and prove its equivalence with $\mathrm{FO}_{\mathcal{L}}$. This algebra is similar to Codd's algebra for finite relations [Cod70], but the operators apply to finite representations of possibly infinite sets. The algebra consists of the following operations: cartesian product, ×, selections ($\sigma_=$, $\sigma_<$, and σ_+), projection, π, set operations (union, $\cup$, intersection, $\cap$, and set difference, $-$), and rename, ρ.

The algebra operations are performed on sets of generalized tuples, i.e. on quantifier-free formulas in DNF. But unlike Kanellakis and Goldin [KG94], we do not assume special encoding for relations and generalized tuples. On the other hand, our algebra can also be viewed as a simplified sublanguage of the algebra of Paredaens, Van den Bussche and Van Gucht [PVV94] (which also includes multiplication).

The algebra will serve as a mere technical tool for the proof of the main theorem. We should note that it has no important preservation property with respect to the size of (the representation of) a database or k-boundedness. However, such properties are not necessary for our purpose. We shall instead use upper bounds on the parameters (size and degree of boundedness) of a database generated by the application of an operation of the algebra (see Section 5 for an in-depth study).

We now define the algebra operators. Suppose R is an n-ary relation represented by a quantifier-free formula, φ, of the form:

$$\varphi \equiv \bigvee_{i=1}^{k} \bigwedge_{j=1}^{\ell} \varphi_{i,j}$$

where the $\varphi_{i,j}$'s are atomic formulas. Then, we also denote the representation φ as a collection of generalized tuples t_i in the set notation:

$$\left\{ t_i \,\middle|\, 1 \leqslant i \leqslant k,\ t_i = \bigwedge_{j=1}^{\ell} \varphi_{i,j} \right\}$$

Furthermore, if I is an instance over signature σ and $R \in \sigma$, we consider the relation $I(R)$ as a set of generalized tuples as above. We also assume that attributes (columns) of relations have names and for each attribute name A, there is a distinct variable x_A associated with it. Attribute names are usually denoted by $A, B, C, \ldots$ (and possibly with subscripts). When the context is clear, we may blur the distinction between variables and attribute names.

Definition 4.1 Let σ be a signature. The family of *algebraic expressions (over σ)* is defined inductively as follows:

1. (R) and $(A : \mathbb{Q})$ are *atomic* expressions, where $R \in \sigma$ is a relation symbol, and A is an attribute name. The set of attributes is the set of attributes of R or $\{A\}$, respectively.

Suppose now that e_1 and e_2 are two algebraic expressions.

2. (Cartesian product) If e_1 and e_2 have disjoint sets of attribute names, then $(e_1 \times e_2)$ is also an expression.
3. (Selection) If F is a selection formula (defined below) involving only attribute names of e_1, then $(\sigma_F\ e_1)$ is an expression. A *selection formula* is an atomic formula of one of the following three forms:

$$t_1 = t_2, \qquad t_1 \leqslant t_2, \qquad t_1 + t_2 = t_3,$$

   where t_1, t_2, t_3 are attribute names or constants (in $\mathbb{Q}$).
4. (Projection) If e_1 has attributes $\{A_1, \ldots, A_n\}$, and moreover $\{B_1, \ldots, B_k\} \subseteq \{A_1, \ldots, A_n\}$, then $(\pi_{B_1,\ldots,B_k}\ e_1)$ [or $(\pi_{\varnothing} e_1)$ if $k = 0$] is an expression.
5. (Set operations) If e_1, e_2 have exactly the same set of attributes, then $(e_1 - e_2)$, $(e_1 \cap e_2)$, and $(e_1 \cup e_2)$ are expressions.
6. (Rename) If A, B are two attribute names and A is an attribute of e_1 but B is not, then $(\rho_{A \to B}\ e_1)$ is also an expression.

We now describe the *semantics* of the algebra. (Note that the operators work directly on generalized tuples, so the semantics is given with respect to generalized tuples.) Suppose that I is an instance of σ, and e is an expression over σ. The *result* of e on I, denoted by $e(I)$, is defined inductively as follows:

1. (a) If $e = (R)$, $e(I) = I(R)$ (a set of generalized tuples).
   (b) If $e = (A : \mathbb{Q})$, $e(I) = \{x_A = x_A\}$, where x_A is the variable corresponding to the attribute A.
2. If $e = (e_1 \times e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in e_2(I)\}$.
3. If $e = (\sigma_F\ e_1)$, $e(I) = \{t \wedge F \mid t \in e_1(I)\}$, where each attribute name A in F is replaced with the corresponding variable x_A.
4. If $e = \pi_{B_1,\ldots,B_k}\ e_1$, then $e(I)$ is obtained from $e_1(I)$ by "eliminating" the variables which do not correspond to attributes B_1 through B_k. One proceeds as follows. Suppose $e_1(I) = \{t_1, \ldots, t_m\}$ and has attributes $A_1, \ldots, A_n$ and $\{C_1, \ldots, C_{n-k}\} = \{A_1, \ldots, A_n\} - \{B_1, \ldots, B_k\}$. We apply the well-known Fourier-Motzkin Elimination method [Sch86] (see below) to eliminate one by one all existentially quantified variables $x_{C_1}, \cdots, x_{C_{n-k}}$ in each of the formulas $\exists x_{C_1} \cdots \exists x_{C_{n-k}} t_i$. Each tuple t_i then results in t'_i. Finally, $e(I) = \{t'_1, \ldots, t'_m\}$.
5. (a) If $e = (e_1 \cup e_2)$, then $e(I) = e_1(I) \cup e_2(I)$.
   (b) If $e = (e_1 \cap e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in e_2(I)\}$.

(c) If $e = (e_1 - e_2)$, then $e(I) = \{t_1 \wedge t_2 \mid t_1 \in e_1(I), t_2 \in (e_2(I))^c\}$, where R^c is the complement of R obtained as follows. Suppose $R = \{t_1, \ldots, t_n\}$ is a set of generalized tuples and for each i, $t_i = \bigwedge\!\!\!\bigwedge_j \varphi_{i,j}$. Then R^c is the formula[9] in DNF which is equivalent to $\bigwedge\!\!\!\bigwedge_i \bigvee\!\!\!\bigvee_j \neg\varphi_{i,j}$.

6. If $e = \rho_{A \to B}\ e_1$, then $e(I) = e_1(I)[x_A/x_B]$ (all occurrences of x_A are replaced by x_B).

The Fourier-Motzkin elimination method (see for instance [Sch86], pp. 155–157) works as follows. Consider a generalized tuple t which defines a polyhedron $P(\overline{x}, y) \subseteq \mathbb{Q}^{n+1}$ described by the inequalities (once the coefficients of y have been normalized):

$$\begin{cases} a^\ell \overline{x} + y \leqslant a_0^\ell & \text{for } \ell = 1, \ldots, L \\ b^k \overline{x} - y \leqslant b_0^k & \text{for } k = 1, \ldots, K \\ c^i \overline{x} \leqslant c_0^i & \text{for } i = 1, \ldots, I \end{cases}$$

where $\overline{x} \in \mathbb{Q}^n$, $y \in \mathbb{Q}$. One can show that after the "elimination" of y (i.e. after P has been projected on its first n coordinates), the relation over $\overline{x}$ is exactly:

$$\left\{ \overline{x} \in \mathbb{Q}^n \;\middle|\; b^k \overline{x} - b_0^k \leqslant a_0^\ell - a^\ell \overline{x} \text{ for all } \ell \text{ and } k,\ c^i \overline{x} \leqslant c_0^i \text{ for all } i \right\}.$$

Therefore:

$$\pi_{\overline{x}}\, t = \bigwedge\!\!\!\bigwedge_{1 \leqslant k \leqslant L, 1 \leqslant \ell \leqslant L} b^k \overline{x} - b_0^k \leqslant a_0^\ell - a^\ell \overline{x} c^i \overline{x} \leqslant c_0^i.$$

It is easy to verify that the algebra, denoted by $\mathrm{ALG}_{\mathcal{L}}$, is equivalent to first-order logic over the class of structures we consider. The proof (which is omitted) is quite similar to that of the equivalence of the classical relational algebra and calculus over finite structures (see [AHV94]).

Theorem 4.1 $\mathrm{FO}_{\mathcal{L}} = \mathrm{ALG}_{\mathcal{L}}$.

We illustrate the above result with the following example.

Example 4.1 Consider the following query over a binary relation R with attributes A, B:

$$\{z \mid \exists x \exists y\ (R(x,y) \wedge y = 2x + z)\}.$$

The equivalent algebra query is:

$$\pi_{A_2} \sigma_{B = A_1 + A_2} \sigma_{A_1 = A + A} \left(R \times (A_1 : \mathbb{Q}) \times (A_2 : \mathbb{Q})\right). \quad \blacksquare$$

[9] Note that the formula may a priori have exponential length in the size of the original formula $\bigwedge\!\!\!\bigwedge \bigvee\!\!\!\bigvee \neg\varphi_{i,j}$. We prove in the next section that it can be done in polynomial length for the families of databases considered here.

Remark. The combination of selections and cartesian products can yield complicated forms of selections. For instance, $\sigma_{B=kA+c}(e)$ (e is an expression with attributes A, B) can be expressed as:

$$\pi_{A,B}\ \sigma_{B=A_{k-1}+c}\sigma_{A_{k-1}=A_{k-2}+A}\cdots\sigma_{A_1=A+A}\left(e\times(A_1:\mathbb{Q})\times\cdots\times(A_{k-1}:\mathbb{Q})\right).$$

Finally, we discuss the complement operation used in defining the semantics for the set difference operation. Computing the complement of a relation R is generally a costly operation. The naive approach to converting a formula $\bigwedge\!\!\bigwedge \bigvee\!\!\bigvee \neg\varphi_{i,j}$ of size n into DNF might generate a formula whose length is exponential in n. However, there are special cases where efficient algorithms exist.

Example 4.2 Suppose R is a set of binary generalized tuples consisting of linear constraints with a fixed set of k distinct (rational) slopes $\alpha_1, \ldots, \alpha_k$. We can view the constraints in R as dividing $\mathbb{Q}^2$ into many "cells" and R as a collection of these cells. Since each cell is a convex polygon, every cell can be defined using at most $2k$ constraints. It can then be verified that there exists a representation of the complement of R, where each tuple involves at most $2k$ constraints. Let S be the set of all possible constraints (involving n variables) in R or obtained from constraints in R by changing the logical predicates and define:

$$U = \left\{ t_1 \wedge \cdots \wedge t_{2k} \;\middle|\; \text{for each } 1 \leqslant i \leqslant 2k,\ t_i \in S \right\}.$$

Then, the complement of R can be defined as:

$$R^c = \left\{ t \in U \;\middle|\; \forall\overline{x}\ t(\overline{x}) \Rightarrow \neg R(\overline{x}) \right\}.$$

Therefore, the complement can be computed in polynomial time for each fixed k. ■

5 Complexity

In this section, we analyze the data complexity of first-order queries. We present two results: (i) a known NC bound [KKR90] in the general case, and (ii) a new AC^0 bound for a restricted class of inputs, namely k-bounded instances for a fixed k. The proof of this last result relies on the algebra introduced in the previous section.

The time (or space) data complexity of a query is the time (resp. space) needed in evaluating the query in terms of the "size" of the representation of the input instances. Formally, we have:

Definition 5.1 Let R be a relation, and ϕ_R its representation over the language $\{=, \leqslant, +\} \cup \mathbb{Z}$. The formula ϕ_R is *of size* $|\phi_R| \leqslant n$ if ϕ_R contains at most n disjuncts (tuples) and at most n distinct constraints, and the absolute values of the integers occurring in ϕ_R are bounded by 2^n (i.e. the absolute values can be represented in binary notation with n bits).

It was shown in [KKR90] that first-order queries with polynomial constraints (over the real numbers) have NC data complexity. This result follows from techniques, first introduced in [BKR86], showing that the theory of real closed fields of fixed dimension (number of variables) can be decided in NC. The same upper bound of course holds in the case of linear constraints.

Theorem 5.1 [BKR86, KKR90] $\mathrm{FO}_{\mathcal{L}}$ is in NC over the class of linear constraint inputs.

We next present the main theorem of this section which applies to a restricted class of inputs that is of practical interest, namely, k-bounded linear constraint inputs. Recall that a k-bounded linear constraint input is a relation that is finitely representable by a quantifier free formula in DNF, such that in each atomic formula occurring in it there are at most k occurrences of the addition symbol, and all constants are integers.

Theorem 5.2 For each (fixed) integer $k \geqslant 0$, $\mathrm{FO}_{\mathcal{L}}$ is in uniform AC^0 over the class of k-bounded linear constraint inputs.

First observe that Theorem 5.2 doesn't carry over for the general case of not k-bounded linear constraint inputs (with binary encoding of natural numbers). Consider a monadic relation containing a single tuple: $R = \{[ax = b \wedge x = b']\}$ for arbitrary values of a, b and b' in $\mathbb{N}$. The boolean query $\pi_{[]}(R) \neq \varnothing$ is true iff $a \times b' = b$. The size of relation R is essentially the size of the three numbers a, b, and b'. Multiplication of numbers in binary notation is not in AC^0 [FSS84]. We can therefore conclude that first-order logic over linear constraint databases is not in AC^0.

Theorem 5.2 extends the now classical result that the relational algebra has AC^0 data complexity over finite structures. Before presenting the proof of Theorem 5.2, we briefly review the proof in the case of the relational algebra over finite structures as it is sketched in [AHV94]. In the case of finite relations, the circuits are constructed uniformly as follows. The gates of the circuit represent pairs of the form $[R, t]$, where R is a relation name (or any algebraic expression, such as $R' \times R''$), and t is a tuple of the same arity as R. The semantics is that the value of a gate $[R, t]$ is 1 iff $R(t)$ holds.

Consider an algebraic query Q. There is a gate of the form $[R, t]$ for each R, either an input relation or an algebraic expression that is a sub-expression of the query Q, and each tuple t which has the proper arity and is built with atomic constants from the input relations. That gives rise to a polynomial number of gates.

The circuit computes the value of $[Q, s]$, for each tuple s of the corresponding arity, starting from the values of the $[R, t]$, where R is an input relation. Most operations are very simple to simulate. For instance, the value of $[R' \times R'', [t', t'']]$ is 1 iff both $[R', t']$ and $[R'', t'']$ have the value 1. The only operation that is slightly more complex is the projection, which requires unbounded fan-in of the OR gates.

In the case of constraint databases, the number of tuples (of atomic values) is infinite. Instead of the tuples, the generalized tuples need to be encoded. We next explain how the encoding is done using gates in a circuit.

Without loss of generality, we make a few assumptions to simplify the presentation. Specifically, we assume that Q is a first-order ***boolean*** query whose input consists of a ***single binary relation*** R. For each natural number n, we exhibit a boolean circuit C_n, of constant depth (depending only upon the query Q and the degree k of boundedness of the inputs) with polynomially many gates in terms of n. The circuit C_n has the property that for each k-bounded input R with a representation ϕ_R of size smaller than n, the circuit C_n, starting on an encoding $enc(\phi_R)$ of ϕ_R, computes an encoding of $Q(R)$. The proof easily extends to inputs with several relations, of arbitrary arities, and to queries with outputs of arity $\geqslant 1$. The circuits then have many output gates, giving an encoding of (a representation of) the output.

The input (under the previous assumptions) is encoded as follows. We first describe how to encode with $3n^3 + 4n^2$ bits any (quantifier free) formula of $\{=, <, +\} \cup \mathbb{Z}$ with two free variables of size n. (i) Integers are encoded in binary notation with n bits. (ii) Constraints of the form $\alpha x + \beta y \Theta \gamma$, where α, β, and γ are integers whose absolute values are smaller than 2^n, and Θ is $=$ or $<$, are encoded on $3n + 4$ bits as follows:

$$\boxed{\overline{\Theta}\ \overline{\alpha}\ |\alpha|\ \overline{\beta}\ |\beta|\ \overline{\gamma}\ |\gamma|}\ ,$$

where the bit $\overline{\Theta} = 0$ (resp. 1) if Θ is $=$ (resp. $<$); the bit $\overline{\alpha} = 1$ (resp. 0) if α is a positive (resp. negative) integer; $|\alpha|$ is the binary representation of the absolute value of α in n bits; and similarly for β and γ.

Since there are at most n constraints in each tuple and at most n tuples in the binary relation R, the whole encoding of a formula for R of size n requires a sequence of $n \times n \times (3n + 4) = 3n^3 + 4n^2$ bits.

During the computation, the syntactic objects encoded in the circuits can grow in size. For instance, bigger integers may result from adding integers of size n. Similarly, constraints over more than two variables are sometimes needed, as a result of an application of the cartesian product, for instance. The cartesian product, along with other operations, also trigger an increase of the number of constraints in each tuple. Therefore, the number of bits allocated to the encoding of integers, constraints, and tuples varies at the different strata (depths) of the circuit. The encoding of bigger integers, constraints over more variables, and tuples containing more constraints, is done in the same manner as above, by adding the required amount of space. Since each first order query can be evaluated using a fixed number of (algebraic) operations, the required additional space can always be figured out once a particular query is given.

In the following we first discuss the projection and set difference operations in the algebra, prove two key lemmas concerning the AC^0 data complexity bound of these two operations, and then present the proof of Theorem 5.2.

The projection operation requires the computation of addition, and repeated addition (bounded multiplication). We first prove that (i) the addition of two integers, and thus (ii) the multiplication of an integer by a given constant can be done in uniform AC^0 with respect to the size of the binary representation of the integers.

Lemma 5.3 The addition of two binary integers of size $\leqslant n$, $a_n a_{n-1} \cdots a_1 a_0$, and $b_n b_{n-1} \cdots b_1 b_0$, can be done by constant-depth circuits with $n+1$ output gates and at most $\mathcal{P}(n)$ gates, where $\mathcal{P}$ is a polynomial.

Proof: Assume that:

$$a_n a_{n-1} \cdots a_1 a_0 + b_n b_{n-1} \cdots b_1 b_0 = c_{n+1} c_n \cdots c_1 c_0$$

The boolean circuit is constructed uniformly with the following formulas:

$$c_0 = \neg(a_0 \Leftrightarrow b_0)$$
$$c_k = (a_k \Leftrightarrow b_k) \Leftrightarrow \bigvee_{i=0}^{i=k-1} \left(\left(\bigwedge_{j=i+1}^{j=k-1} (a_j \vee b_j) \right) \wedge (a_i \wedge b_i) \right)$$

for each $1 \leqslant k \leqslant n$, and

$$c_{n+1} = \bigvee_{i=0}^{i=n} \left(\left(\bigwedge_{j=i+1}^{j=n} (a_j \vee b_j) \right) \wedge (a_i \wedge b_i) \right)$$

where "$\Leftrightarrow$" is an abbreviation for a circuit of depth 3 using only $\neg$, $\wedge$, and $\vee$ nodes: $x \Leftrightarrow y \equiv (x \wedge y) \vee (\neg x \wedge \neg y)$. The depth of the circuit is 7, and the number of nodes is $O(n^3)$. ■

Remark. On the other hand, addition of rational numbers (encoded as pairs of natural numbers) is not in AC^0. Indeed, this would imply that multiplication of natural numbers is also in AC^0. Indeed, consider $\frac{1}{x} + \frac{1}{y} = \frac{y+x}{y \times x}$. As a consequence, our proof does not carry over to the case where databases are defined with rational numbers as parameters. This follows from the fact that addition of parameters coming from the input constraints is required to compute an application of projection.

We next prove that the projection can be done in AC^0. More precisely, we prove that for each tuple, there is a circuit of fixed depth, with a polynomial number of gates, that computes the projected tuple.

Lemma 5.4 Let S be a k-bounded set of linear constraints over n variables $x_1, ..., x_n$ for some k, and i a positive integer $\leqslant n$. The projection $\Pi(S)$ of S on variables $\{x_1, ..., x_n\} - \{x_i\}$ is computable in AC^0.

In Lemma 5.4, the set S denotes a single k-bounded tuple, i.e. the total number of occurrences of the addition symbol in each constraint in S is bounded by k. It follows easily that the projection of an entire k-bounded relation can be done in AC^0. The circuit contains essentially copies of the circuit that computes the projection individually for each tuple.

Proof of Lemma 5.4: The AC^0 upper bound for the projection of a tuple relies on the following simple technical claim, which shows how addition and multiplication are used in the computation of the resulting constraints after a set of constraints has been projected onto some components.

Claim: Let S be a k-bounded set of linear constraints over n variables $x_1, ..., x_n$ of the form: $\sum_{\ell=1}^{n} \alpha_\ell x_\ell \, \Theta \, \alpha_0$, and let $\mathcal{C}(S)$ be the set of variable coefficients (namely, α_ℓ for each $\ell \geqslant 1$), and $\mathcal{C}_0(S)$ the set of constant coefficients (namely, α_0), in the constraints of S. Let Π be the projection on variables $\{x_1, ..., x_n\} - \{x_i\}$ for some i. Then the following holds:

- The variable coefficients of $\Pi(S)$ are obtained by additions and multiplications of variable coefficients in $\mathcal{C}(S)$, and
- The constant coefficients of $\Pi(S)$ are obtained by multiplications of a constant coefficient in $\mathcal{C}_0(S)$ with a variable coefficient in $\mathcal{C}(S)$, and additions.

The proof of the claim is rather straightforward. Consider the following two constraints in S:

$$\sum_{\ell=1}^{n} \alpha_\ell x_\ell \leqslant \alpha_0 \qquad \text{and} \qquad \sum_{\ell=1}^{n} \alpha'_\ell x_\ell \geqslant \alpha'_0$$

where $\alpha_i > 0$ and $\alpha'_i > 0$. The resulting constraint using the Fourier-Motzkin method is:

$$\sum_{\ell=1}^{n} (\alpha_\ell \alpha'_i - \alpha_i \alpha'_\ell)\, x_\ell \leqslant (\alpha_0 \alpha'_i - \alpha_i \alpha'_0).$$

Note that in the above constraint the coefficient for x_i is 0 (hence x_i is eliminated). The new constraint verifies the statement of the claim. It is easy to see that for any type of linear constraints the claim holds.

We now see that the projection of S can be done in AC^0. Since S is a k-bounded set of linear constraints, the variable coefficients are not larger than the constant k. Therefore, the resulting constraints are obtained by multiplication with a constant (integer) not larger than k, and by addition. These two operations can be done in AC^0 (it follows from Lemma 5.3). Moreover, the number of resulting new constraints is at most quadratic in the number of initial constraints, using the Fourier-Motzkin method. ■

The only other operation that requires some care is the set difference. The next lemma is devoted to the complement operation, that can be used to define set difference.

Lemma 5.5 Let k be a (fixed) positive integer and $\mathcal{K}_k$ be the class of k-bounded linear constraint relations. There is a polynomial function $\mathcal{P}$, such that for each relation R in $\mathcal{K}_k$ of size n, the following conditions hold for the complement, R^c, of R: (i) $|R^c| \leqslant \mathcal{P}(n)$, and (ii) R^c is computable in AC^0 in the size of R.

Proof: Assume that R is an r-ary relation of size n. Since R is a k-bounded linear constraint relation, it follows that the number of different slopes of hyperplanes in R is the number of nonnegative integer solutions to the equations $z_1 + z_2 + \ldots + z_r = j$ where $1 \leqslant j \leqslant k + 2$. In particular, $k' = \sum_{j=1}^{k+2} \binom{r-1+j}{j}$. (When $r < k$, $k' \leqslant O(k^r)$.) Therefore each cell (in the sense of [Col75]) can be defined by a tuple with no more than $2k'$ constraints. Assume that R is defined with $\ell \leqslant n$ different constraints. It can be seen that the constraints needed to define the cells in the complement R^c are the existing constraints, and their variants obtained by replacing the predicate in each constraint with one of "=", "<", or ">". This generates at most 3ℓ constraints. Since no other constraint is required, every cell in the plane is therefore definable with at most $2k'$ constraints. There are 3ℓ possibilities for each constraint, thus it leads to at most $(3\ell)^{2k'}$ possible cells. The number of cells is therefore bounded by a polynomial function in n (see also [Col75]), and the complement can easily be computed in AC^0 (using only operations in ALG$_{\mathcal{L}}$ as shown in Example 4.2). ■

We are now ready to prove Theorem 5.2.

Proof of Theorem 5.2: The proof is by induction on the structure of the formula expressing the query. We can always assume that the boolean query is of the form $\pi_\varnothing e$, where e is some algebraic expression (i.e. a test of emptiness).

Basis: Assume $e = R$. To verify that $\pi_\varnothing R$ is false, it suffices to check that each tuple in R defines an empty set. This can be done by applying the Fourier-Motzkin method. It follows from Lemma 5.4, that this can be done in AC^0.

In the sequel, we prove by induction that we can compute in AC^0, an encoding of $e(R)$ starting from an encoding of R for each subexpression e.

Induction: The induction step depends on the last algebraic operations performed. We first consider the gates of the circuit, and then illustrate how it is wired. We next establish upper bounds on (i) the number of constraints in each tuple, and (ii) the number of tuples in the new relations, resulting from the application of algebraic operations. Let e_i be a k-bounded relation of n_i tuples, each tuple consisting of k_i constraints ($i = 1, 2$). Note that both the number n_i of tuples and the number k_i of constraints may not be exact. However they are upper bounds. Physically, the circuits contain the space to encode n_i tuples of k_i constraints.

1. If $e = (e_1 \times e_2)$, then e is a k-bounded relation containing $n_1 \times n_2$ tuples, each of which is represented with $k_1 + k_2$ constraints.

2. If $e = (\sigma_F\ e_1)$, then e is a k-bounded relation containing n_1 tuples, each of which is represented with $k_1 + 1$ constraints.
3. If $e = \pi\ e_1$, where π just eliminates a single variable, then e is a k^2-bounded relation containing n_1 tuples, each of which is represented with at most k_1^2 constraints.
4. If $e = (e_1 \cup e_2)$, then e is a k-bounded relation containing $n_1 + n_2$ tuples, each of which is represented with $max(k_1, k_2)$ constraints.
5. If $e = (e_1 \cap e_2)$, then e is a k-bounded relation containing $n_1 \times n_2$ tuples, each of which is represented with $k_1 + k_2$ constraints.
6. If $e = (e_1 - e_2)$, then e is a k-bounded relation containing $\mathcal{P}(n_1, n_2, k_1, k_2)$ tuples, each of which is represented with at most $\mathcal{P}'(k_1, k_2)$ constraints. In the binary case (e_1 and e_2 binary), $\mathcal{P}(n_1, n_2, k_1, k_2) = n_1 \times (3n_2k_2)^{2k'^2}$ and $\mathcal{P}'(k_1, k_2) = max(k_1, 2k'^2)$, where $k' = \dfrac{(k+3)(k+4)}{2} - 1$. For larger arities, both the number of tuples and the number of constraints per tuple are bounded by similar polynomials.

The above follows from the definition of the algebraic operations in Section 4, from Lemma 5.4 for the case of the projection, and from Lemma 5.5 for the case of the set difference. In this last case, $(e_1 - e_2) = (e_1 \cap e_2^c)$, where e_2^c is a k-bounded relation containing, in the binary case, a maximum of $(3n_2k_2)^{2k'^2}$ tuples, each having at most k'^2 constraints.

It follows that the number of tuples and the number of constraints in each tuple are bounded by some polynomial function. Note that the integers occurring in the constraints during the computation, come either from the input, from the query, or result from a projection. One projection generates quadratic numbers, and so their binary representation has twice the initial space. Therefore, the size of integers is linear in n. For each algebraic sub-expression of the query Q, and each tuple t of the adequate form obtained as described above, we associate a series of gates encoding the pair Q, t in the circuit. Other gates are also required in computing the additions of constants for the new constraints resulting from the projection operator. As shown in Lemma 5.4, there are only a polynomial amount of these. The selection also requires built-in gates to encode the constraint in the selection itself. This is easily done with a number of gates bounded by a constant in the size of the query. Essentially, no more gates are needed to encode the whole circuit. It follows that the number of gates needed to encode the whole computation is also polynomially bounded.

We now see how the wires between the gates previously presented, can be uniformly defined.

The algebraic operations have various effects on their inputs. They can modify (or rearrange) the initial tuples and/or the constraints. (i) The union operation changes only at the relation level and the initial tuples remain unchanged. (ii) Cartesian product, selection, and intersection create new tuples from old tuples, by using the initial constraints which are not changed. (iii) Set difference creates new constraints obtained from old constraints, by just changing the predicates in the constraints. (iv) Projection creates new constraints, with new parameters as shown in Lemma 5.4.

In the case of $\cup, \cap, \times$, and σ_F, it is clear that the new tuples can be computed easily in AC^0. More precisely, these operations result only in a reorganization of existing constraints inside the tuples, and of tuples inside the new relations. The wires are essentially used to copy values (with no computation). They do not have to be materialized.

Two operators, projection and set difference, deserve a more thorough examination. Indeed, they result in the definition of new constraints, with new parameters obtained from old parameters by addition, or iterated addition. It follows from Lemmas 5.4 and 5.5, that the two operations can be computed in uniform AC^0. ■

Kanellakis and Goldin [KG95] suggested to study the data complexity of first-order queries over linear constraint databases in the case where integers are encoded in unary. In the remainder of the section, we briefly discuss the data complexity of $FO_{\mathcal{L}}$ for arbitrary linear constraint databases, i.e. without the restriction of being k-bounded. We show that under the unary representation of integers, the data complexity remains in AC^0.

Let m be the circuit input size. The *unary representation* of an integer $n \leqslant m$ is a string $a_m a_{m-1} \cdots a_2 a_1$ where $a_i = 1$ for each $1 \leqslant i \leqslant n$ and $a_i = 0$ for $n < i \leqslant m$. We now show that the addition and the multiplication of two integers encoded in unary representation can be done by boolean circuits of constant depth (i.e. in AC^0).

Theorem 5.6 Let $m \in \mathbb{N}$. The addition (and multiplication) of two (positive) integers I_a, I_b such that $I_a + I_b \leqslant m$ (resp. $I_a \times I_b \leqslant m$) in unary representation, $a_m a_{m-1} \cdots a_2 a_1$ and $b_m b_{m-1} \cdots b_2 b_1$, can be done by constant-depth circuits with m output gates and polynomially (in m) many gates.

Proof: We first consider addition. Let $I_c = I_a + I_b$ and $c_m c_{m-1} \cdots c_2 c_1$ be the unary representation of I_c. It is observed that I_c can be computed by counting all gates from $a_m a_{m-1} \cdots a_2 a_1$ and $b_m b_{m-1} \cdots b_2 b_1$ that are true. Indeed, for each $1 \leqslant i \leqslant m$, c_i can be defined by the following boolean function f_i (note that $a_j = 1 \Rightarrow a_\ell = 1$ for each $\ell \leqslant j$):

$$f_i = a_i \vee \left(\bigvee_{j=1}^{i-1} (a_{i-j} \wedge b_j) \right) \vee b_i$$

It is easy to see that f_i can be realized by a circuit of depth no more than 2 and of no more than $(2m-1)$ gates.

Now let $I_c = I_a \times I_b$ be the product and we assume again $c_m c_{m-1} \cdots c_2 c_1$ is the unary representation of I_c. Then, the multiplication can be viewed as the following sum of integers in unary representation:

$$\sum_{1 \leqslant i \leqslant m, b_i = 1} a_m a_{m-1} \cdots a_2 a_1$$

Thus, it is easy to see that for each $1 \leqslant i \leqslant m$, c_i is defined by the following boolean function g_i:

$$g_i = \bigvee_{\eta(j,k)} (a_j \wedge b_k)$$

where the condition $\eta(j,k)$ states that $1 \leqslant j, k \leqslant m$, $j \times k \geqslant i$, and both $(j-1) \times k$ and $j \times (k-1)$ are $< i$. Hence, the circuit realizing g_i has depth 2 and number of gates linear in m. Note that for each $m \in \mathrm{N}$, the circuits $f_1, ..., f_m, g_1, ..., g_m$ can be uniformly constructed. Therefore, addition and multiplication of integers in unary representation are in AC^0. ■

Since both addition and multiplication of integers in unary representation can be computed by circuits of constant depth, it can be verified that the data complexity of $\mathrm{FO}_{\mathcal{L}}$ over linear constraint databases remains in AC^0 under the unary encoding assumption. The size of the numbers that are derived by a query from the numbers in the input is defined by a polynomial which depends only upon the query itself, and enough space is devoted to them in the circuit. The proof follows the same lines as the proof of Theorem 5.2. The assumption of k-boundedness was needed in Theorem 5.2 to prove that projection involved only multiplication by a constant. This assumption is not needed here since multiplication of unary numbers can be done in AC^0. For the set difference operation, using multiplications it is possible to "triangulate" the plane (when the arity is 2) or hyperplane (when the arity is higher) using the constraints in the input. Thus to compute the complement, one needs to consider only tuples with up to a fixed number (depending only on the arity) of constraints (3 constraints when the arity is 2). We can use an approach for computing the complement and set difference similar to the one described in the proof of Lemma 5.5 and the k-boundedness assumption is not necessary.

In the next section, we examine consequences of the complexity upper bound on the expressive power of linear constraints.

6 Expressive Power

In this section, we study the expressive power of first order query languages for linear constraint databases. In particular we consider queries from relational database theory (parity), graph theory (graph connectivity), and geometry (region connectivity) and show that these queries are not first order expressible. The proof of these results uses the AC^0 upper bound on data complexity (Theorem 5.2) and first order reductions from boolean functions, such as PARITY which is known to be outside AC^0 [FSS84].

Let $\sigma = \{R\}$ be a signature where R is a unary relation symbol. For a database instance I of σ, the *parity* query answers "yes" if $I(R)$ is finite and has an even cardinality. The *graph connectivity* query is defined over a signature consisting of a single binary relation G. The query answers "yes" on an instance I if $I(G)$ is a connected finite graph. For the third example, we consider the *k-dimensional region connectivity* query over possibly infinite input instances, where $k \geqslant 1$. The query is also a boolean query and answers "yes" on an instance I if every pair of points in $I(R)$ can be linked by a continuous curve lying entirely in $I(R)$. Note that for $k = 1$, the query can be easily expressed.

Theorem 6.1 The following queries are not definable in $\{=, \leqslant, +\} \cup \mathbb{Q}$:

1. Parity of cardinality,
2. Graph connectivity,
3. k-dimensional region connectivity for each $k \geqslant 2$.

Proof: By Theorem 5.2, it is sufficient to show that these queries are not in AC^0. We first consider the parity query and describe a straightforward reduction from the boolean function PARITY. The PARITY function takes n boolean inputs and returns "true" if the number of inputs equal to 1 is even. Now let $x_1, \ldots, x_n$ be the n inputs for PARITY. We construct a database I over the signature with one unary relation symbol R as follows: $I(R) = \{i \mid x_i = 1\}$. Clearly the database is definable using only equality constraints, without the addition symbol. In other words, I is in $\mathcal{K}_0$ (0-bounded). Obviously, the construction can be done in first order and $\text{PARITY}(x_1, ..., x_n) = 1$ iff the parity query on I answers "yes". For graph connectivity, we use the classical reduction from the parity query. Let I be an input instance of the parity query and G be a binary relation symbol. Suppose $I(R) = \{a_1, ..., a_n\}$, and, without loss of generality, $a_1 < a_2 < \cdots < a_n$. We define an instance J over G as follows. Let $J(G)$ be the symmetric closure of the set $\{(a_1, a_n)\} \cup \{(a_i, a_{i+2}) \mid 1 \leqslant i \leqslant n-2\}$. It is easy to verify that parity on I answers "yes" iff $J(G)$ is connected. Finally, for region connectivity in dimension $k \geqslant 2$, it is shown in [GS95] that it is not in AC^0, by a reduction from the boolean function MAJORITY. ■

The previous result can be generalized to various contexts.

Corollary 6.2 The queries of Theorem 6.1 are not definable with linear constraints over the following domains: the natural numbers, $\mathbb{N}$, the integers, $\mathbb{Z}$, the rationals, $\mathbb{Q}$, and the reals, $\mathbb{R}$.

Acknowledgment

The authors thank Paris Kanellakis and Dina Goldin for their comments on an earlier version of the paper and their suggestion to examine data complexity under unary encoding.

References

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1994.

[Ban78] F. Bancilhon. On the completeness of query languages for relational data bases. In *Proc. 7th Symp. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 112–123. Springer-Verlag, 1978.

[Bea76] D. R. Bean. Recursive Euler and Hamilton paths. In *Proc. American Mathematical Society*, volume 55, pages 385–394, 1976.

[BKR86] M. Ben-Or, D. Kozen, and J. Reif. The complexity of elementary algebra and geometry. *Journal of Computer and System Sciences*, 32(2):251–264, April 1986.

[CH80] A. K. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–78, 1980.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *Communications of ACM*, 13:6:377–387, 1970.

[Col75] G. E. Collins. Quantifier elimination for real closed fields by cylindric decompositions. In *Proc. 2nd GI Conf. Automata Theory and Formal Languages*, volume 35 of *Lecture Notes in Computer Science*, pages 134–83. Springer-Verlag, 1975.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17:13–27, 1984.

[GS94] S. Grumbach and J. Su. Finitely representable databases (extended abstract). In *Proc. 13th ACM Symp. on Principles of Database Systems*, 1994.

[GS95] S. Grumbach and J. Su. Finitely representable databases, 1995. Full version of [GS94], invited to *JCSS* (Special Issue of PODS '94).

[Har91] D. Harel. Hamiltonian paths in infinite graphs. *Israel Journal of Mathematics*, 76:317–336, 1991.

[HH93] T. Hirst and D. Harel. Completeness results for recursive data bases. In *Proc. 12th ACM Symp. on Principles of Database Systems*, pages 244–252, 1993.

[HH94] T. Hirst and D. Harel. Recursive model theory, 1994. Draft.

[Joh90] D. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, Elsevier-North Holland, 1990.

[KG94] P. C. Kanellakis and D. Q. Goldin. Constraint programming and database query languages. In *Proc. 2nd Conference on Theoretical Aspects of Computer Software (TACS)*, April 1994. (To appear in a LNCS volume, Springer-Verlag).

[KG95] P. C. Kanellakis and D. Q. Goldin. Personal communication, 1995.

[KKR90] P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 299–313, Nashville, 1990.

[Mos57] A. Mostowski. On recursive models of formalized arithmetics. *Bulletin de l'Académie Polonaise des Sciences, III*, 5:705–710, 1957.

[Par78] J. Paredaens. On the expressive power of the relational algebra. *Information Processing Letters*, 7(2):107–111, February 1978.

[PVV94] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proc. 13th ACM Symp. on Principles of Database Systems*, pages 279–88, 1994.

[PVV95] J. Paredaens, J. Van den Bussche, and D. Van Gucht. First-order Queries on Finite Structures over the Reals. In *Proc. 10th IEEE Symp. on Logic in Computer Science*, to appear.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, Chichester, 1986.

[Va94] J. Väänänen. Personal communication.

[Vau60] R. L. Vaught. Sentences true in all constructive models. *Journal of Symbolic Logic*, 25(1):39–53, March 1960.

A Constant-Space Sequential Model of Computation for First-Order Logic

(preliminary draft)

Steven Lindell†

Department of Computer Science, Haverford College, Haverford, PA 19041-1392

Abstract. We define and justify a natural sequential model of computation with a constant amount of read/write work space, despite unlimited (polynomial) access to read-only input and write-only output. The model is both deterministic, uniform, and sequential. The constant work space is modeled by a finite number of destructive read boolean variables, assignable by formulas over the canonical boolean operations. We then show that computation on this model is equivalent to expressibility in first-order logic, giving a duality between (read-once) constant-space serial algorithms and constant-time parallel algorithms.

§ 0 Introduction

Summary

Problems computable in constant time on a uniform parallel model of computation (a type of PRAM) have been elegantly characterized as those expressible in first-order logic (FO) on binary strings [I]. It is also known that FO is identified with LH, the logtime alternation hierarchy based on random-access Turing machines [BIS]. We provide an additional correspondence between FO and those problems computable in constant space on a deterministic sequential model of computation.

The key to this idea is a very careful measuring of work space in a machine. Ordinarily, read-only input and write-only output are not considered part of the read/write work space. This is an essential concept for defining the complexity class L (logarithmic-space). We go somewhat farther in our model, and do not include in the work space any storage mechanism required to access the input or output, be it memory addressing or tape scanning. By making the access scheme oblivious, we are careful not to let the machine cheat by using the memory addresses or head positions as read/write storage.

Furthermore, we take the additional step of separating the flow of control of the machine from the computation it is performing. Specifically, we imagine a machine controlled by a simple programming language with: a finite set of read/write boolean variables; the operations AND, OR, NOT; composition of program statements; and a strict form of definite loops. No conditionals are allowed in the programming language (if...then, or while ... repeat) to insure the oblivious nature of the computation. In addition, we impose a read-once (destructive read) condition that prevents a read/write boolean variable from being read more than once without an intervening write. No such restriction applies to the input or output however.

† Partially supported by NSF grant CCR-9403447, and the John C. Whitehead faculty research fund at Haverford College. Electronic mail address: <slindell@haverford.edu>.

The following is representative of our main theorem:

Theorem: *A query on binary strings is first-order definable if and only if it is computable by an constant-space read-once serial algorithm.*

Motivation

Classically, when defining sub-linear space on a Turing machine, we resort to an off-line model which separates read-only input and write-only output from the read/write work tape. In this way we can get robust definitions of SPACE(log n) and above. And although logspace transducers have proved to be a useful reducibility between problems, a finer notion of reduction based on first-order translations has led to some illuminating results: a very restricted version of the Berman-Hartmanis conjecture [ABI]; and a very deep result concerning the recursive enumerability of the polytime queries [D']. It has also been shown that first-order logic provides a robust notion of uniformity for the study of the fine structure of low-level circuit complexity classes [BIS], and the corresponding first-order reduction has been shown equivalent in [AG] to a much earlier notion of logspace rudimentary reductions. Furthermore, first-order translations are based on the classical notion of interpretation as spelled out in [E], and serve as excellent reductions which preserve the completeness of well-known NP-problems [D] as well as newer ones (the boolean formula value problem being complete for ALOGTIME) [B].

Finite-state transducers fail in this capacity because they cannot do the arithmetic needed to convert binary input from one simple form into another (like reversing a string), nor can they provide polynomial magnification (required for the existence of complete problems). This happens in any standard off-line model with space below $O(\log n)$.

Our sequential model of computation is able to lift these limitations while still operating under a form of constant-space constraint. This is because our model permits multiple unidirectional heads which can re-scan the tape, but forbids head movement which is non-oblivious. We will measure the actual amount of read/write work space in a very careful fashion. In particular, our model does not even count any space used to access the input, whether it be a head scanning a tape, or an address into random-access memory.† The reasoning for this is intuitive: if the input (output) tape is read-only (write-only), and the access to the tape is oblivious, then the machine cannot use the tape head as a read/write storage mechanism, so that space does not count. Clearly this intuition, if correct, extends to any fixed number of heads. If access to the tape is not oblivious, and we allow 2-way arbitrary movement of the head in a finite automata, then all of logspace is achieved ([G], p. 53). In fact, our model will be explained in terms of a programming language, much the same as the presentation of primitive recursion in [ibid, p.20], and can be compared with the uniform constant-width circuits of [BI, section 6], which gives a corresponding characterization of uniform-NC^1. Also, [C] has given machine-independent algebraic characterizations of AC^0 and various other small complexity classes using sequential operations.

Overview

Section 1 provides a brief review of first-order definable queries, including examples. Of particular importance is the discussion of binary string structures and the special numerical predicates for arithmetic. Section 2 defines and justifies the constant-space model of sequential computation that this paper introduces, comparing it with the classical definition of finite-state automata. Section 3 illustrates this model with two contrasting bit-serial examples: parity; and addition. Section 4 contains the main result, mentioned above, and its proof. The remainder of the paper, section 5, concludes by describing possible directions for future research.

† However, it will be preferable to use a cursor to mark head position, since this method of memory access appears aesthetically more sequential, as opposed to RAM which involves an implicit use of parallelism (address decoding).

§ 1 Background

First-order Queries

One way of mathematically studying the computational complexity of combinatorial problems is to directly examine how difficult it is to define them. Instead of measuring asymptotic resources (such as time or space) required to compute a problem, one can classify problems as to the power of the logic required to express their solution. Specifically, an input instance is a *finite relational structure*,

$$\langle A, R_1, \ldots, R_k, c_1, \ldots, c_l \rangle$$

consisting of a finite set A, called the *domain*, together with *relations* R_i (each of a specified arity on A), and *constants* c_j (individual elements of A). The output is determined by a *query*, a global relation across structures of the same type (signature), mapping each one to a relation (of fixed-arity) on the structure. One of the simplest "languages" for expressing queries is that of *first-order* logic. Formulas in first-order logic permit: individual variables interpreted as ranging over the domain; constant symbols for each constant c_j, predicate symbols for each relation R_i and equality (=); the Boolean connectives $\wedge$, $\vee$, and $\neg$; and quantification of the variables. We use FO to denote the class of all queries determined by such first-order formulas. For further background, see [E].

Simplicity Example

One of the easiest and most familiar examples of finite structures is the class of directed graphs — all structures having a binary edge relation E over a finite domain of vertices V:

$$G = \langle V, E \rangle \qquad E \subseteq V^2$$

The problem of determining if a graph is *simple* (no self-loops, all edges undirected) is an example of a graph property, or boolean query of arity 0 which is expressible as a first-order sentence.

$$G = \langle V, E \rangle \text{ is } \textit{simple} \text{ iff } G \models \theta \quad \text{where}$$

$$\theta \equiv \neg(\exists x)[E(x, x)] \wedge (\forall y)(\forall z)[E(y, z) \rightarrow E(z, y)]$$

The first part of θ says that no vertex has an edge to itself, and the second part says that if there is an edge from one vertex to another, then there must be an edge going the opposite direction.

Ordering Example

Another example is the problem of determining if a binary relation constitutes a total linear *order* of the vertices. Using more suggestive notation

$$B = \langle A, < \rangle \text{ is } \textit{ordered} \text{ iff } B \models \psi$$

where ψ is the conjunction of the following universally quantified axioms

$\neg(x < x)$ (irreflexivity)
$(x \neq y) \rightarrow [(x < y) \vee (y < x)]$ (totality)
$[(x < y) \wedge (y < z)] \rightarrow (x < z)$ (transitivity)

Binary String Structures

Ordering the Positions

Although graphs are universal (as general as any other type of structures) for expressibility purposes [L1], it seems necessary to work on ordered structures to express computations, particularly the contents of the input as it is presented to a machine. Modern digital computers use binary strings for I/O, and we can represent these in the form $B = \langle\{0, 1, ..., n-1\}, <, U\rangle$, where the domain is the set of positions in the string, < orders these positions from left to right in the usual fashion $0 < 1 < ... < n-1$, and U indicates where the 1's and 0's are by true and false, respectively. For instance, the binary string 1010 is represented by the structure $\langle\{0, 1, 2, 3\}, <, \{0, 2\}\rangle$, with $0 < 1 < 2 < 3$.

● < ○ < ● < ○
0 1 2 3

In the figure, closed circles indicate where U is true, and open circles where U is false.

In general, given a binary string $w \in \{0, 1\}^*$, we create a canonical structure for it:

$$\langle |w|, <, \{i : w_i = 1\}\rangle,$$

where $|w|$ is the length of w, and w_i is the i^{th} symbol of w.

Adding Arithmetic

To accurately capture fine notions in resource-bounded computation (such as parallel time) appears to require, in addition to the ordering, a method whereby the binary input to a machine can be accessed effectively [I]. For this purpose (what might be called address arithmetic) it suffices to have a special predicate which, for each domain element i, gives the location of 1's and 0's in its binary representation:

$$\text{bit}(i, j)$$
$$\Leftrightarrow$$
$$\text{the } j^{\text{th}} \text{ position in the binary representation of } i \text{ is a } 1.^{\dagger}$$

For instance, bit(5, 1) is false, since $5 = (101)_2$, and there's a 0 in the first position (the rightmost bit is treated as the zeroeth position). This leads us to the following definition.

Definition: For each $w \in \{0, 1\}^*$, define the *binary string structure* for w to be:

$$\mathrm{A}_w = \langle |w|, <, \text{bit}, \{i : w_i = 1\}\rangle.$$

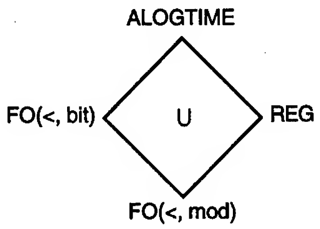
We distinguish the correspondingly augmented class of first-order queries by the notation FO(<, bit), indicating that < and bit are assumed to be "givens" in the same way = is taken for granted. See [L2] for a brief discussion that this in fact is equivalent to first-order logic with arithmetic on the domain, FO(+, *, ^).

† For a discussion of the logical importance of the bit predicate, see [L2].

§ 2 The Machine Model

Comparison with Regular Languages

Before discussing the machine model, it will be instructive to compare the computational complexity of FO(<, bit) with the more familiar regular languages, denoted REG. First note that both FO(<, bit) and REG are strictly contained in ALOGTIME (= uniform-NC^1). A classical result is that the regular languages, viewed as collections of binary string structures without bit, are precisely those definable in monadic second-order logic (see [S]). Equally important for our purposes is the fact that first-order logic on binary string structures w/o bit corresponds exactly to the star-free fragment of REG. In fact, FO(<, bit) $\cap$ REG is equal to the class of first-order definable queries on binary strings with order, together with the unary numerical predicates $M_k = \{m{\cdot}k : m = 0, 1,\ldots\}$ for each $k > 0$ [BCST]. This is denoted FO(<, mod) in the following strict containment diagram.



Two simple examples serve to illustrate the contrast between FO(<, bit) and REG.

$$PARITY = \{w \in \{0, 1\}^* : w \text{ has an even number of ones}\}$$
$$MIDPOINT = \{0^n1^n : n = 0, 1, 2\ldots\}$$

A trivial finite automaton can recognize *PARITY*. Yet [FSS] show that *PARITY* cannot be definable in FO(—) even with arbitrary numerical predicates. Conversely, a trivial FO(<, bit) formula can express *MIDPOINT* (by using addition). However, *MIDPOINT* is the classic example of a non-regular language. For comparison purposes, note that

$$EQUAL = \{w \in \{0, 1\}^* : w \text{ has an equal number of zeros and ones}\}$$

is neither in REG nor FO(<, bit), but is in ALOGTIME.

Comparison with Finite Automata

To help explain the discrepancies between regular languages and first-order logic, we turn to the table below which shows the two distinct differences between finite automata and the sequential deterministic model we propose.

	finite automata	*proposed model*
input/output access	single oblivious scan	multiple oblivious passes
flow of control	state machine	restricted state machine

Multiple Heads

Whereas a finite automata scans its input precisely once from left to right, our model allows for multiple heads, each of which is permitted to re-scan the input tape. To prevent positional information from being used as read/write storage, we restrict their movements to be oblivious (unidirectional with reset to the left edge) so that their locations depend only on the length of the input. Moreover, these heads serve not only to access the input, but also as read-only clocks which can measure execution time, and we include a mechanism whereby their relative and absolute positions can be queried.

Destructive Read

When a finite state machine is implemented in hardware, flip-flops are used to store the current state, while boolean gates combine this information with the current input bit in an interconnection network to set the next state which is stored back in the same flip-flops. A fixed number of gates and binary storage elements assure a constant-space resource bound. In our model, gate types are restricted to AND, OR, NOT, and we insist that flip-flops are destructively read (making computations more akin to iterated boolean formulas).

Definition of Model

While it is certainly possible to continue in this fashion and define our model in terms of time clocks and circuit diagrams, it is more convenient to describe our model in terms of a programming language, to make serial algorithms textually representable. By restricting storage to (read-once) boolean variables, the software will naturally constrain the model to a constant amount of (destructive-read) space. By limiting the constructs to composition and a strict form of definite loops (indexed by tape heads), oblivious flow of control and input access will be guaranteed.

We take a multi-head machine M consisting of an input tape together with a fixed number of heads to scan the tape. It is equipped with head-crossing detectors which keep track of the relative left-right positions of any pair of heads, and (resettable) binary counters which keep track of the absolute position of each head. A mechanism is built-in whereby any particular bit of a counter can be queried.

The machine itself is controlled by a sequential program P, whose syntax assures that the machine obeys the polynomial-time and constant-space resource bounds, and whose semantics assure the read-once restriction and oblivious head movement. We define these very simple programs by induction.

Booleans

The basic data type is a boolean and only boolean variables are available for read-write storage. At any point in time, a *boolean variable* is either in the *read* or *unread* state. Apart from variables, there are other boolean values directly accessible in the machine model which do not have read restrictions. These are called *direct values*, which come from the input (using any tape head h as index), comparing the positions of any two tape heads, or querying any counter bit of a tape head. They are summarized in the list below.

`b`	a read-once/write boolean variable
`I[h]`	*false* if h is currently positioned over a zero on the tape, *true* otherwise
`i < j`	a comparison test which yields *true* iff head i is to the left of head j
`bit(i, j)`	*true* iff the jth column in the binary counter for head i is set[1]

When combining these, only the operations of AND, OR, NOT are allowed in forming boolean expressions.

[1] Purists may object to this because the standard implementation of a binary counter requires more than $O(1)$ operations per "cycle", though the number of operations amortized over an entire loop is $O(1)$. In any event, this is similar to the read-only clock in section 6 of [BI].

Assignments

If b is a boolean variable, and e is a boolean expression all of whose boolean variables are distinct and unread, then

```
    b := e;
```

assigns the value of e to b. Every boolean variable in e becomes read, and the status of b becomes unread (regardless of the state it was in before). Taken together, these conditions syntactically prevent a boolean variable from being read twice without an intervening write. Just note that this *read-once condition* can also be syntactically enforced in the further constructions below, even though we do not indicate precisely how it is done (essentially, keep track of which variables are required to be unread upon entering a block, and which are unread upon exiting a block).

Composition

If P and Q are programs, then so is their sequential composition,

```
    P;
    Q;
```

provided the read-once condition is not violated.

Loops

Tape heads can serve as guarded parametrized controls for a loop. If the read-once condition is not violated, then the looping construct

```
    LOOP h
      P;
```

binds tape head h to scan the input tape from the first cell to the last, performing one iteration of P for each such position $0, \ldots, n-1$, while moving h on the tape from the beginning (left) to the end (right). The position of h is not allowed to change inside P (assume for simplicity that loop heads are not re-bound, i.e. not nested with the same name). Also, assume that h returns to the left edge after completing the loop.

Acceptance

By designating one of the boolean variables as the *answer*, we can define string acceptance.

Definition: Let P be a read-once constant-space sequential program as described above. We say[2] $\langle P, \underline{h}\rangle$ *accepts* w if running program P on input w with initial head positions (range: $\{0, \ldots, |w| - 1\}$) starting at $\underline{h}$ results in a true answer. Omission of any or all of the positions $\underline{h}$ implies those heads begin at the left end of the tape (position zero).

The corresponding language determined by P is defined in the usual way.

[2] We use underline to notate tuples.

§ 3 Examples

We illustrate constant-space programs by two serial algorithms to provide both a counterexample and example to the read-once condition.

Parity

This simple program computes the parity of an input string $a(1)...a(n)$.

```
p := false                  {parity initially zero}
LOOP h                            {from LSB to MSB}
  p := a(h) XOR p                   {exclusive-or}
```

But the boolean variable p violates the read-once condition because it must be read twice when forming the exclusive-or from the canonical base of boolean operations (although $a(h)$ may be read any number of times since it is a read-only input).

Addition

Contrast this with the schoolbook algorithm for serial binary addition of two n-bit numbers, $a(n)...a(1) + b(n)...b(1)$, which yields an n-bit sum $s(n)...s(1)$ and a carry. It requires a single read/write boolean variable c for the carry:

```
c := 0                                    {initialize carry}
LOOP h                                     {from LSB to MSB}
  s(h) := a(h) XOR b(h) XOR c                    {see notes}
  c := a(h) AND b(h) OR c AND (a(h) OR b(h))
```

Note how a simple re-parenthesization, in the last line, of the 3-input majority function (which is all the carry really is) results in a program obeying the read-once condition. Also, note well that the reads of c used in assigning $s(h)$ do not count since the destination is write-only. (We have not included output statements in this preliminary draft of the paper).

A consequence of our main theorem will be that this constant-space serial algorithm implies the existence of a constant-time parallel algorithm for n-bit binary addition, which I find quite surprising by itself, since the standard carry look-ahead algorithm was not a completely trivial observation in its time. Conversely, the existence of a constant-time parallel algorithm for binary addition implies the existence of a (read-once) constant-space serial algorithm. This phenomenon of time-space duality will be discussed further in § 5.

§ 4 Main Result

Theorem: A binary language $L \subseteq \{0, 1\}^*$ is recognized by a read-once constant-space sequential program P if and only if it is definable by a first-order sentence ϕ over the class of binary string structures. I.e. $w \in L$ iff $A_w \models \phi$.

Proof: ($\Leftarrow$) We show by induction over the quantifier depth d of a first-order $\{<, bit, U\}$-formula $\varphi(\underline{x})$ in prenex form, that there is a program P such that

$$A_w \models \varphi[\underline{h}] \quad \Leftrightarrow \quad \langle P, \underline{h} \rangle \text{ accepts } w$$

where $\underline{h}$ is a tuple of numbers between 0 and $n-1$, whose length equals the length of $\underline{x}$.

Basis

If $\varphi(\underline{x})$ is quantifier-free, then it is easy to see that the boolean formula determined by $\varphi[\underline{h}]$ can be computed by a loop-free program, since for all i and j in $\underline{h}$, the atomics $i < j$ and bit(i, j) are built-in direct values of the same name in the machine model, and because $U(h)$ can be directly read off the input by `I[h]`, since the tape head assigned to h starts on that square to begin with by assumption.

Induction

Suppose $\varphi(\underline{x}) \equiv (Qy)\psi(y, \underline{x})$, where $Q \in \{\exists, \forall\}$. By induction hypothesis, $\psi[k, \underline{h}]$ is computed by a program $\langle P, (k, \underline{h})\rangle$ To compute $\varphi[\underline{h}]$, we loop the first head around the program P, and add an additional variable to store and compute the result of the quantification. Here is the program for existential quantification:

```
b := false
LOOP k
  P;
  b := b OR (the answer from P)
answer b
```

Note that program P must be repeatedly run for each position k. A similar dual program can be used for universal quantification.

($\Rightarrow$) The reverse direction is considerably more difficult.[3] We will express a read-once program P by a collection of first-order formulas, $\Pi = \{\pi_b(\underline{x}) : b \in P\}$ (but from now on dropping the clarification $b \in P$ since it will be obvious from context), such that for each boolean variable b of P (left unread upon exit), the final result left in b after running P with initial head positions $\underline{h}$ on input w is the same as the truth value of $A_w \models \pi_b[\underline{h}]$. Boolean variables will appear as nullary atomic relations in these formulas, and furthermore, we will guarantee that each one occurs at most once in Π, calling this the *single occurrence property*.

Taking advantage of the definability of arithmetic in FO(<, bit), we will freely use addition and multiplication for computations in the formulas we construct. As before, initial head positions will correspond to free variables in formulas. Once these are fixed, a program P can be thought of as a map from booleans to booleans. Our proof continues by syntactic induction on P.

Base Case: P is a single assignment statement: `b := e;`.

In this case, e is a boolean expression of direct values and boolean variables. Hence P can be represented quite easily by the single formula $\pi_b \equiv e$, where the comparison and binary calculation of head positions are by definition the atomic relations < and bit, and any references to input values `I[h]` are replaced by the atomic $U(h)$. It remains to check that π_b satisfies the single occurrence property, but this is clearly the case since each boolean variable occurs at most once in e in order for P to satisfy the read-once condition.

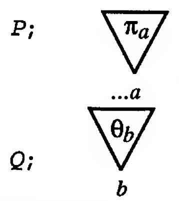
Inductive Steps:

Consider the program `P; Q;`. By induction hypothesis, there is a collection of first-order formulas $\Pi = \{\pi_b\}$ expressing P, and a similar collection $\Theta = \{\theta_b\}$ expressing Q, each satisfying the single occurrence property. If a boolean variable a of Q does not appear

[3] Even more difficult than we will make it appear. See the coda at the end of this proof for an explanation.

in P, then adjoin to Π the identity formula $\pi_a \equiv a$. Similarly, if a boolean variable b of P does not appear in Q, then adjoin $\theta_b \equiv b$ to Θ. Note that neither of these changes affect the read-once condition of P or Q, nor the single occurrence property for Π or Θ.

Let $\theta_b[a \leftarrow \pi_a]$ denote for each boolean variable a in θ_b, the substitution of formula π_a. Now, we claim that the *composition* $\Pi \circ \Theta = \{\theta_b[a \leftarrow \pi_a]\}$ expresses `P;  Q;`.



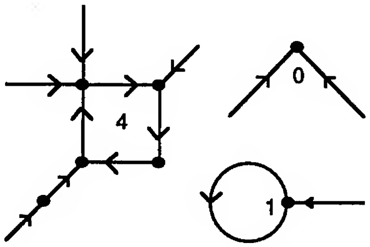
syntax tree for $\theta_b[a \leftarrow \pi_a]$

Moreover, the single occurrence property for Θ guarantees that each formula π_a in Π is used exactly once to replace an occurrence of the boolean variable a in Θ. Since all the original boolean variables occurring in Θ are replaced in this manner, the only remaining boolean variables in $\Pi \circ \Theta$ are those occurring in Π. Since each formula π_a in Π was used exactly once, and since Π satisfies the single occurrence property, we can see that $\Pi \circ \Theta$ satisfies the single occurrence property too.

Next, consider the program `LOOP h  P;` which is by far the most difficult part of the argument. By induction hypothesis, there is a collection of first-order formulas $\Pi(x) = \{\pi_b(x)\}$ with the single occurrence property, such that the value of b after executing P is $\pi_b[h]$, for h equal to the position of the loop variable `h` (other unbound head positions have been omitted for clarity).

First remove negations by rewriting P. Push all negations to the bottom, then replace every occurrence of `NOT b` by a new boolean variable `b'`. Now, follow each assignment `b := e;` by `b':= NOT e;`, where again we push negations to the bottom. Since every boolean variable appearing in e occurs once, both a boolean variable and its negation cannot both be in e, so the new program still satisfies the read once condition.

The single occurrence property insures that the *syntactic dependency graph* of Π, defined as the graph whose vertices are boolean variables in Π, and whose edges are given by $\{(a, b) : a \text{ appears in } \pi_b\}$ for all boolean variables a and b, has out-degree ≤ 1. This means it looks like a bunch of disjoint whirlpools.



syntactic dependency graph

Since boolean variables in disjoint components are independent of each other, then without loss of generality it suffices by syntactic decomposition to consider a program P with only a single component. We now confine our discussion to such a case, and define the *order* of P to be the length m of the cycle that appears (see numbers in figure), reserving 0 for the case when there is a root instead of a cycle. And define the *height* of P to be the length h of a longest path from any node to its closest node on the cycle (or root if there is none). Variables on the cycle are termed *recursive*, and variables off the cycle are said to be of *finite dependency*.

The plan is to break up the n iterations of the loop into three sections, for n (equal to the length of the input) sufficiently large. The first section, called the 'leader', consists of an initial run of h iterations. This is followed by the 'main loop', which repeats an m-iterate block a total of $l = (n - h)$ div m times. This in turn is followed by a 'trailer' of the remaining $(n - h)$ mod m iterations.

Leader

As long as $n \geq h$, the composition

$$\Lambda = \Pi[0] \circ \ldots \circ \Pi[h-1]$$

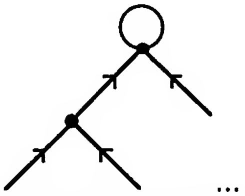
is clearly first-order expressible. Furthermore, since each boolean variable b of finite dependency has height less than or equal to h, each corresponding $\lambda_b \in \Lambda$ will have no boolean variables occurring in it, and we call such formulas *explicit*. Since Λ is a finite power of Π under composition, the collection $\Lambda = \{\lambda_b\}$ including the recursive boolean variables b still satisfies the single occurrence property.

Main Loop

We now proceed with the important task of contracting the cycle. If $m = 0$, there is no cycle to contract, it is easy to see that the dependency graph is a tree, and hence each boolean variable has finite dependency. In this case, each boolean variable will have an explicit formula to describe its value, essentially consisting of the last h iterations of the loop. The details can be determined by continuing the proof for the case $m > 0$ and just skipping the parts which deal with a recursive boolean variable. On the other hand, if $m \geq 1$, then we simplify things by dividing the remaining $n - h$ iterations into $l = (n - h)$ div m blocks of size m, writing each such block as

$$\Theta[j] = \Pi[h + j \cdot m] \circ \ldots \circ \Pi[h + j \cdot m + m - 1] \quad \text{for} \quad j = 0, \ldots l - 1$$

It needs to be noted that starting the loop variable at h and incrementing it in multiples of m is first-order expressible using arithmetic, and that the dependency graph for Θ satisfies the single occurrence property (being a fixed power of Π). The purpose of all this is to obtain a dependency graph for Θ with components of order 1, so that each recursive boolean variable is in its own cycle.



one component of the syntactic dependency graph for Θ

The main loop consists of composing $\Theta[j]$ from $j = 0$ to $l - 1$. Coming into the main loop, all λ_b for b non-recursive are explicit formulas (from the leader). We shall demonstrate that the partial iterates defined by

$$\Psi[k] \Leftrightarrow \Lambda \circ \Theta[0] \circ \Theta[1] \circ \ldots \circ \Theta[k-1] \ \text{for} \ \ 0 \le k \le l$$

can be expressed by a collection of first-order formulas $\Psi(x) = \{\psi_b(x)\}$, with the single occurrence property. Intuitively, $\psi_b[k]$ will express the value of b before going into $\Theta[k]$ (after $h + k{\cdot}m$ iterations of P) and the final value of b after completing the main loop will be given by $\psi_b[l]$.

We first derive the formulas $\psi_b(x)$ for the values of non-recursive boolean variables b at each stage of the main loop, by induction on their height (the longest path from a leaf). Let $\theta_b(x)$ be the first-order formula for b in $\Theta(x)$.

$$\psi_b(x) \Leftrightarrow (x = 0) \wedge \lambda_b \vee (x > 0) \wedge \theta_b(x-1)[a \leftarrow \psi_a(x-1)]$$

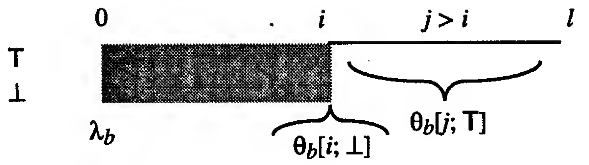
If b is a leaf in Θ, then θ_b contains no boolean variables, and the indicated substitution is vacuous. However, if b is not a leaf, then the replacements ψ_a have already been expressed by induction hypothesis (each a occurring in θ_b is by definition of smaller height). Since λ_b contains no boolean variables, it follows by induction that neither does ψ_b, since as long as b is not recursive, each a is not either.

The recursive variables are the most interesting part. Let b be a recursive boolean variable, and let $\theta_b(x) \in \Theta$. The only occurrence of a recursive boolean variable in $\theta_b(x)$ is a single positive occurrence of b itself, by virtue of the fact that the dependency graph for Θ has order 1, and because we have eliminated negations while preserving the read-once condition. In particular, this makes $\theta_b(x)$ monotone in b. So b is true at the end of the main loop just in case there is a last stage i in which b is or becomes true (here is where we require monotonicity) and remains true at all subsequent stages $j > i$. In explaining this we'll use $\mathbf{T}$ to stand for *true*, $\perp$ for *false*, and $\theta_b[i;\, t]$ to stand for $\theta_b[i][b \leftarrow t]$, where t is a boolean formula. For b to be or become true at stage $i > 0$ means $\theta_b[i;\, \perp]$, or in the case of $i = 0$, $\theta_b[0;\ \lambda_b]$, where b enters the main loop with value λ_b that it obtains from the leader.[†] To make things simpler, we can combine these two cases into the single formula $\theta_b[i;\, i = 0 \wedge \lambda_b]$. For b to remain true at stage $j > i$ means $\theta_b[j;\, \mathbf{T}]$, which can be included in a further combination as $\theta_b[j;\, j > i \vee j = 0 \wedge \lambda_b]$ for $j \ge i$. Of course, the values of the other (non-recursive) boolean variables a that θ_b may depend on must be known at every stage j of the main loop in order for this to work. But recall that we have already ascertained these formulas $\psi_a[j]$, so we can substitute those occurrences out. The value of b at the end of the main loop is given by:

$$\psi_b[l] \equiv (\exists x.0 \le x < l)(\forall y.x \le y < l)\theta_b(y)[b \leftarrow y > x \vee y = 0 \wedge \lambda_b;\ a \leftarrow \psi_a(y)]$$

where a ranges over all (non-recursive) boolean variables appearing in θ_b. To help understand this, the following picture graphs the boolean value of b (shown on the vertical axis) through time (as shown by j on the horizontal axis) in the case where $\psi_b[l]$ ends up true.

[†] It doesn't matter if b is already true at point i, since monotonicity insures that $\theta_b[i;\, \perp] \Rightarrow \theta_b[i;\, \mathbf{T}]$.



By construction, the special clause $j > i \vee j = 0 \wedge \lambda_b$ evaluates to the correct boolean value in all cases. To see that the resulting collection Ψ satisfies the single occurrence property, note that for each non-recursive boolean variable, ψ_a contains no boolean variables, as we observed earlier. For the remaining recursive boolean variables, recall that the collection $\Lambda = \{\lambda_b\}$ satisfies the single occurrence property. Now observe that Ψ and $\Lambda \circ \Theta$ have the same dependency graph by examination of the constructed formulas. Since both Λ and Θ have the single occurrence property, their composition must also.

Trailer

The remaining iterations (anywhere from 0 to $m - 1$) can be expressed by

$$\mathrm{T} = \Pi[n - ((n - h) \bmod m)] \circ \ldots \circ \Pi[n - 1]$$

The arithmetic involved is certainly first-order definable as a function of n, which also determines the length of the composition. However, the astute reader will notice that we would not be able to maintain the single occurrence property for T if we were to just combine the m possible cases implied above into a single formula. But over all n, there are only m possible syntactic dependency graphs for T. For each of those possibilities, T has the single occurrence property guaranteed by composition, and we can use $\Psi \circ \mathrm{T}$ to express the semantics of the loop.

Coda

As a matter of fact, we see from the trailer that the end result is parametrized by n over finitely many choices. So technically speaking , it is necessary to stipulate from the beginning that each program P is represented by *one of a finite number of collections of first-order formulas* (each individually satisfying the single occurrence property). The proof is then complete only by observing that this technical construction can be carried forward through the proof without affectation, the important thing being that the number of possibilities grows with each application of the induction hypothesis, but remains bounded and first-order dependent only on n. The final result of a boolean variable in a program is then expressed as a boolean combination of these finitely many cases.

If perchance n is too small to carry out the whole above construction, just enumerate the finitely many cases involved.

§ 5 Directions

Improvements

In the model we have defined, the use of a binary counter to monitor absolute head position is somewhat inelegant, and the looping construct to bind head movement is somewhat restrictive. It would be much nicer if the heads could be controlled by `MOVE` instructions, and loops replaced by `DO` *h* `TIMES` ... for some head position *h*. The lack of conditionals would retain obliviousness, and these instructions alone would be sufficient to obtain the necessary arithmetic, without having to keep track of relative and absolute head positions. Unfortunately, this may compute too much, and the success of this more aesthetic approach seems to depend on resolving a certain fundamental question in finite model theory, namely: does *iteration* (as a logical construct) over finite initial segments of the natural numbers with successor close at FO(+, *)? Surprisingly, iterated multiplication (exponentiation, ^) is in FO(+, *). However, a more promising approach which doesn't rely on complexity theoretic separations might be to just devise some simple mechanism of tying head movements to a global system clock using, for example, frequency dividers.

Extensions

Other Complexity Classes

It is relatively easy to obtain a sequential deterministic characterization of ALOGTIME (uniform-NC^1) by dropping the destructive read restriction in our constant-space model (in fact the proof would be much easier, cf. [BI]). This identifies ALOGTIME in a reasonably elegant manner with constant-space serial algorithms. To my mind the real challenge is to find a similar sequential deterministic model for uniform constant-depth threshold circuits (TC^0). Presumably, the model might use integer variables with the read-once restriction. It would also be instructive to directly prove the containment $TC^0 \subseteq NC^1$ in this setting, demonstrating a concrete constant-space serial algorithm for counting.

Time/Space Duality

We observed in section 3 that the standard schoolbook algorithm for binary addition is a read-once constant-space serial algorithm. If you go through the proof of our main theorem, you will obtain a first-order formula for it which can be seen to be virtually the same as the classic carry look-ahead method:

$$\begin{aligned} &s(i) = a(i) \oplus b(i) \oplus c(i) \qquad \text{where,} \\ &c(i) = (\exists j)[j < i \wedge a(i) \wedge b(i) \wedge (\forall k)[j < k < i \rightarrow (a(i) \vee b(i))]] \end{aligned}$$

Perhaps the best-known (and probably oldest) parallel algorithm, it says that there is a carry into a column if and only if some previous column generated one, and every subsequent column propagated it.

A somewhat surprising consequence of our work is that in general, for polynomial bounds on size and length, destructive read constant-space serial algorithms are equivalent to constant-time parallel algorithms. Classical parallel-time / serial-space duality is a phenomenon that appears only to extend down to resource bounds of $O(\log n)$ [H]. However, the seemingly peculiar read-once restriction has allowed us to take this duality all the way down to $O(1)$ time or space. By modifying our model to allow dynamically allocated new storage (additional boolean variables), it should be possible to use Immerman's iterated first-order formulas to extend our results to provide a tight correspondence between read-once space and quantifier-depth. One particularly intriguing possibility is a duality theorem for TC^0, involving the as yet undiscovered sequential model mentioned above.

Multiplication

Since n-bit multiplication is in TC^0, it is instructive to consider the usual schoolbook algorithm for binary multiplication as an example of a serial algorithm with integer variables.

```
            d_{n-1} ... d_0
          x e_{n-1} ... e_0
  -----------------
  = p_{2n-1} ...... p_0
```

The partial products (not pre-computed) are added right to left in columns, and the partial sums for each column accumulated top to bottom, with a multi-bit carry into the next column.

```
s := 0                            {initialize partial sum}
LOOP i FROM 0 TO 2*n-1                {go from LSB to MSB}
  LOOP j FROM 0 TO i                  {sum for ith column}
    s := s + d(j) * e(i-j)           {add next term in sum}
  p(i) := s mod 2                           {product bit}
  s := s div 2                       {carry to next column}
```

Notice that this uses variable loop bounds and the read/write integer variable *s* of $O(\log n)$ bits. Even though *s* appears to be read twice, the operations of incrementing and halving, if combined, are read-once at the bit level. It is intriguing to wonder if a parallel dual to this serial algorithm might not provide a simplified witness to the fact that multiplication can be performed in uniform-TC^0. Unlike carry look-ahead, the current parallel constructions are rather more complex and involved than this simple and intuitive serial method [IL].

Acknowledgments. I am grateful for discussions with Dave Mix Barrington and Sorin Istrail, who introduced me to the equivalence between constant-depth circuits and read-once constant-width branching programs [IZ]. I also appreciate discussions with Sam Buss, Dejan Zivkovic, Steve Tate, Eric Allender, Howard Straubing, Ken Regan, Scott Weinstein, Peter Clote, and Ian Parberry (who made the particularly good suggestion of using tape heads instead of the random-access method I was using). I welcome further comments, questions, and suggestions. Please contact me at my electronic mail address.

References

[ABI] E. Allender, Balcázar, N. Immerman "A First-Order Isomorphism Theorem" to appear in SIAM Journal on Computing. A preliminary version appeared in Proc. 10th Symposium on Theoretical Aspects of Computer Science, Springer-Verlag LNCS 665, pp. 163-174, 1993.

[AG] E. Allender, V. Gore "Rudimentary reductions revisited" Information Processing Letters **40** 89-95 (1991).

[B] S. Buss, "Algorithms for Boolean Formula Evaluation and for Tree Contraction" in *Arithmetic, Proof Theory, and Computational Complexity*, editors: Peter Clote and Jan Krajícˇek, Oxford University Press, pp.95-115, 1993.

[BCST] D. Mix Barrington, K. Compton, H. Straubing, D. Thérien "Regular Languages in *NC*^{1}" JCSS, June 1992 pp. 478-499.

[BI] D. Mix Barrington, N. Immerman "Time, Hardware, and Uniformity" IEEE Structures, 1994 pp.176-185.

[BIS] D. Mix Barrington, N. Immerman, H. Straubing "On Uniformity in NC^1" JCSS **41**, pp.274-306 (1990).

[C] P. Clote "Sequential, machine-independent characterizations of the parallel complexity classes AlogTIME, AC^k, NC^k and NC." in *Feasible Mathematics*, S. Buss and P. Scott editors, Birkhäuser 1990.

[D] E. Dahlhaus, "Reduction to NP-complete problems by interpretations" LNCS 171, Springer-Verlag, pp.357-365, 1984.

[D'] A. Dawar "Generalized Quantifiers and Logical Reducibilities" Journal of Logic and Computation, Vol 5, No. 2, pp. 213-226, 1995.

[E] H. Enderton, *A Mathematical Introduction to Logic*, Academic Press, 1972.

[FSS] M. Furst, J.B. Saxe, M. Sipser "Parity, Circuits, and the Polynomial-time Hierarchy" Math. Syst. Theory **17**, pp. 13-27, 1984.

[G] Y. Gurevich "Logic and the Challenge of Computer Science" in *Trends in Theoretical Computer Science*, Editor: Egon Börger, Computer Science Press, 1988, pp.1-57.

[H] J. W. Hong *Computation: Computability, Similarity, and Duality* Wiley 1986.

[I] N. Immerman, "Expressibility and Parallel Complexity" SIAM Journal of Computing vol. 18 no. 3, June 1989, pp. 625-638.

[IL] N. Immerman, S. Landau "The Complexity of Iterated Multiplication" Information and Computation **116**(1):103-116, January 1995.

[IZ] S. Istrail, D. Zivkovic "Bounded-width polynomial-size Boolean formulas compute exactly those functions in AC^0" Information Processing Letters **50**, pp.211-216, 1994.

[L1] S. Lindell, The Logical Complexity of Queries on Unordered Graphs, Ph.D. Dissertation, UCLA 1987.

[L2] S. Lindell, "A Purely Logical Characterization of Circuit Uniformity" IEEE Structure in Complexity Theory (1992) pp.185-192.

[S] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*, Birkhäuser, 1994.

Logics Capturing Relativized Complexity Classes Uniformly

J.A. Makowsky[1]* and Y.B. Pnueli[2]**

[1] Department of Computer Science, Technion–Israel Institute of Technology
Haifa, Israel
(janos@cs.technion.ac.il)
[2] Institut für Informatik, Freie Universität Berlin
Berlin, Germany
(yachin@inf.fu-berlin.de)

Revised version, March 1995

Abstract. We introduce the notion of a logic capturing a relativized complexity class uniformly by treating both generalized quantifiers and oracles as indeterminates and requiring that the correspondence be uniform. Besides reinterpreting previous results from this point of view, we show that Fixed Point Logic with inflationary fixed points $IFPL$ captures **P** uniformly, whereas First Order Logic $FOL[K_1, \ldots, K_m]$ with any number of generalized quantifiers cannot capture uniformly a relativized complexity class which contains $\mathbf{NC}_2$. This contrasts the fact that in the non–uniform approach both $IFPL$ and $FOL[ATC]$ capture **P**.

1 Introduction and Survey

In this paper we continue our investigations, cf. [MP93, MP94], of the relationship between logics augmented with generalized quantifiers and oracle computations. Intuitively, we think of a logic as a descriptive language and of generalized quantifiers as library calls. The underlying logic allows us to compose such library calls into more complex programs. Similarly, we can think of Oracle Turing machines as computing devices with oracle calls. Our purpose is to match logics augmented with generalized quantifiers with classes of Oracle Turing machines (Relativized Complexity Classes) in a uniform way.

This gives us not only a reinterpretation of previous results, but also a deeper insight into the mechanism of how logics capture complexity classes. It also allows us to distinguish the power of the underlying logic from the power of particularly chosen generalized quantifiers. When this distinction is not made, every logic can be viewed as being First Order Logic augmented by a set of

* Partially supported by a Grant of the French-Israeli Binational Foundation (1994), a Grant of the German-Israeli Foundation (1995) and by the Fund for Promotion of Research of the Technion–Israeli Institute of Technology.

** Minerva Fellow 1993-1994.

generalized quantifiers. In particular, First order Logic, Fixed Point Logics and Second Order Logic appear on the same level, cf. [MP93]. For example, the complexity class **P** (Polynomial Time) is captured by $FOL[ATC]$ (First Order Logic with Alternating Transitive Closure), [Imm87] and also by $MFPL$ (Fixed Point Logic with Monotone Fixed Points), [Var82]. The logic $IFPL$ (Fixed Point Logic with Inflationary Fixed Points) has the same expressive power as $MFPL$, as shown in [GS86, Lei90]. By introducing our distinction, we can exhibit the intrinsic difference between these two approaches: $FOL[ATC, Q]$ will be different from $IFPL[Q]$, when Q is a generalized quantifier treated as a variable. Once spelled out, this is not surprising, but illuminating.

Outline of the paper

We assume the reader is familiar with the basics of generalized quantifiers and logics as described in [EFT80, MP93, BF85] and with logics capturing complexity classes, as in [Imm87, Imm88, Imm89].

In section 2 we introduce our notion of *uniformly capturing* of relativized complexity classes.

In this section we give also a detailed outline of our results. In section 3 we discuss various interpretations of our results and their impact on complexity theory. In this section we also explain and correct an imprecision in [MP94].

In section 4 we give the exact definitions of our uniform approach to model checking and introduce the *query dependency*. This is used to formulate and prove our main technical lemma 32. It is also used to prove upper bounds on uniform model checkers.

In section 5 we define a relativized (or oracle) language, $L_0(K)$, which is a modified version of an oracle language used by J. Buss [Bus88, Bus86].

In section 6 we determine the exact complexity of $L_0(K)$.

In section 7 we use this oracle language to give a proof of our main theorems 11, 31, 34.

In section 8 we draw our conclusions and formulate some conjectures.

Acknowledgments

We have benefited from conversations with participants of the LCC conference (first author) and the Freiburg Logic Seminar (second author), where we presented our results. In particular, We would like to thank A. Dawar, G. Gottlob, T. Imhof, and D. Leivant. whose comments have considerably improved our paper. In particular, subsection 3.2 has been almost literally taken from comments sent to us by A. Dawar, and G. Gottlob has clarified the role of the oracle computation model in theorem 7, proposition 8 and theorem 9.

2 Logics Capturing a Relativized Complexity Class Uniformly

In this section we introduce our notion of *logics capturing a relativized complexity class uniformly*. We view formulas of logics with generalized quantifiers as type–2 objects (depending on the interpretation of the quantifiers). Similarly, we view oracle languages as type–2 objects (depending on the choice of the concrete oracle). We call the latter both *relativized* and *oracle complexity classes*, depending the aspect we wish to stress.

2.1 Oracle Turing Machines and Languages

Definition 1 ([GJ79]). An *Oracle using Turing Machine* (OTM) is a tuple $\{Q, \Gamma, \delta\}$ where Q is a finite set of states, Γ is the tape alphabet and δ is a transition function.

From this definition it is clear that the description (as opposed to the behaviour) of an OTM is independent of the specific oracle set used. We make this distinction explicit in the following

Definition 2 (Oracle Languages). Let $\mathbf{D}$ be a class of Turing machines with some time and space bounds (a complexity class).

(i) $OTM(\mathbf{D}(K))$, the set of Oracle Turing Machines with oracle parameter K, which are in $\mathbf{D}$.
(ii) $L(M(K))$, for $M(K) \in OTM(\mathbf{D}(K))$, the language accepted by $M(K)$.
(iii) $\mathbf{D}^K$, the set of languages accepted by Oracle Turing Machines in $OTM(\mathbf{D}(K))$.
(iv) We denote by $Lang(A)$ the set of languages over the alphabet A. Let $L : Lang(A) \rightarrow Lang(A)$ a function and $K \in Lang(A)$. We denote by $L(K)$ the languages obtained by applying L to K. The family of languages $L(K)$ obtained uniformly using L are called *oracle languages*.
(v) An oracle language $L(K)$ is said to be *uniformly recognized by* $M \in OTM(\mathbf{D}(K)$ if for every language K $M(K)$ recognizes $L(K)$.

Definition 3 (Uniformly Capturing). Let $\mathcal{L}[Q]$ be a logic with a distinguished generalized quantifier and $\mathbf{D}^K$ an oracle complexity class. We say that $\mathcal{L}[Q]$ *captures* $\mathbf{D}^K$ *uniformly* if there are functions

(i) $tr_1 : \mathcal{L}[Q] \rightarrow OTM(\mathbf{D}(K))$ such that for every oracle K and for every formula $\phi \in \mathcal{L}[Q]$ the translation of ϕ, $tr_1(\phi)$, is a model checker for $\phi(Q_K)$; and
(ii) $tr_2 : OTM(\mathbf{D}(K)) \rightarrow \mathcal{L}[Q]$ such that for every oracle K and for every oracle Turing machine $M(K) \in OTM(\mathbf{D}(K))$, the models of the translation of $M(K)$ are exactly the strings accepted by $M(K)$, i.e. $Mod(tr_2(M(K))) = L(M(K))$.

2.2 Positive Results

In [MP94] we have shown the following:

Proposition 4 (Makowsky–Pnueli). *We denote by $u(\infty)$ the restriction of Buss' unbounded oracle computation model, [Bus88] to bounded stacks, as introduced in [MP94].*

(i) For $\mathbf{L}^K$ with the oracle model $u(\infty)$ $FOL[DTC, K]$ captures $\mathbf{L}^K$ uniformly.
(ii) For $\mathbf{NL}^K$ with the oracle model $u(\infty)$ $FOL[TC, K]$ captures $\mathbf{NL}^K$ uniformly.
(iii) For $\mathbf{AL}^K$ with the oracle model $u(\infty)$ $FOL[ATC, K]$ captures $\mathbf{AL}^K$ uniformly.

Although $\mathbf{AL} = \mathbf{P}$, we shall see below that for the $u(\infty)$ oracle computation model there are oracles K such that $\mathbf{AL}^K$ may be different from $\mathbf{P}^K$.

To capture $\mathbf{P}^K$ uniformly we look first at Fixed Point Logic with inflationary fixed points ($IFPL$). This choice allows the introduction of generalized quantifiers without technical problems. It is rather straight forward to see that, using the standard oracle consultation model of Ladner and Lynch, [LL76], we have the following:

Theorem 5. *For $\mathbf{P}^K$ with the standard oracle model, $IFPL[K]$ captures $\mathbf{P}^K$ uniformly.*

Proof. We have to write down the behaviour of the OTM in $IFPL$. The proof follows closely the presentation in [AHV94] or [EF95]. The main difference is, that in the description of the transition table of the Turing Machine we need the general quantifier for the oracle consultation and the formulas in the scope of the general quantifier contain further application of the fixed point operator for decoding the contents of the oracle tape.

We could also look at the case of Fixed Point Logic where the variable of the fixed point operator occurs only positively in the formula (MFPL). In the absence of generalized quantifiers this garantees monotonicity. $IFPL$ and $MFPL$ both capture $\mathbf{P}$ and their intertranslatability was shown in [GS86] and simplified in [Lei90]. The notion of capturing $\mathbf{P}$ uniformly is not well defined for $MFPL$. We have to restrict ourselves to oracles K, for which we know that positive occurrence of the variables still ensures monotonicity. We call such oracles K *monotone*.

With this definition it is easy to show:

Theorem 6. *For $\mathbf{P}^K$ with the standard (unbounded) oracle model, $MFPL[K]$ captures $\mathbf{P}^K$ uniformly for monotone oracles.*

However, it is not clear whether the intertranslatability of $IFPL[K]$ and $MFPL[K]$ holds uniformly for monotone K.

Looking at Fixed Point Logic in an unrestricted way ($PFPL$) the choice of the oracle consultation model is again crucial. G. Gottlob has pointed out that for the oracle computation model introduced by I. Simon, [Sim77], which is equivalent to the bounded model of Buss, [Bus88], one can show:

Theorem 7 (Gottlob). *For* **PSpace**^{K} *with the bounded oracle model, $PFPL[K]$ captures* **PSpace**^{K} *uniformly.*

Remark. If we use SOL, Second Order Logic, rather than Fixed Point Logic, we get similar results for the polynomial hierarchy **PH**, [MP95].

Remark. A Theorem similar to Theorem 6 was also noticed by I. Stewart in [Ste94]. Stewart showed that $MFPL[HAM]$, monotone fixed point logic augmented with the quantifier expressing Hamiltonicity, captures $\mathbf{P}^{\mathbf{NP}}$. However, he does not address the issue of replacing HAM by other oracles, and therefore the question of uniformity does not arise. Also G. Gottlob, [Got95a, Got95b], deals with related questions concerning logics capturing relativized logspace complexity classes, where the base logic is First Order Logic FOL. It is clear, that FOL, in contrast to $FOL[DTC]$ (4), cannot capture **L** uniformly. We return to the question of what FOL could capture uniformly in section 8.

2.3 Negative Results

Now we discuss negative results. The first is an observation of G. Gottlob concerning the choice of oracle consultation models in the case of **PSpace**. In contrast to theorem 7 with the bounded oracle consultation model, under the standard oracle consultation model we have:

Proposition 8 (Gottlob).

(i) **NEXPTime** $\subseteq$ **PSpace**$^{\mathbf{NP}}$, *but*
(ii) Every formula of $PFPL[K]$ has a model checker in **PSpace** *for oracles $K \in$* **NP**.

Proof. (i) **NEXPTime** $\subseteq$ **PSpace**$^N P$: This inclusion is due to the fact that under the standard oracle model, i.e., the Ladner-Lynch model, the oracle string is not subject to any size-bound. This means that the **PSpace** machine may write exponential strings to its oracle-tape. The oracle is **NP**, but with exponential input it behaves like **NEXPTime** w.r.t. the overall OTM input string. Thus, any problem in **NEXPTime** can be solved by a **PSpace**$^N P$ machine under the standard oracle model.

(ii): Consider partial fixpoint logic $PFPL$. Assume $PFPL$ is enriched by a generalized quantifier Q_K corresponding to some oracle $K \in$ **NP**. Then, given the fixed arities of the signature, any formula $\phi(\bar{x}) \in PFPL[K]$, where $\bar{x}$ denotes a list of free variables, evaluates to a *polynomially* sized relation for each finite structure $\mathcal{A}$. Thus any formula $Q_K\bar{x}\phi(\bar{x})$ corresponds to an application of the quantifier Q_K to a polynomial relation, or, equivalently to an oracle-call of oracle K fed with a polynomial size string. Thus, in $PFPL[K]$, there is no way of feeding an **NP**-oracle with an exponential string, and actually, since $K \in$ **NP**, it holds that, by abuse of notation, $PFPL[K] = PFPL = PSPACE$.

Thus we get, in contrast to theorem.7

Theorem 9 (Gottlob). *Assume* **NEXPTime** $\neq$ **PSpace**. *Then under the standard oracle consultation model of Ladner and Lynch $PFPL$ cannot capture* **PSpace** *uniformly.*

To understand better the difference between logics uniformly capturing a complexity class, and the usual notion of logics capturing a complexity class, let us look at the following

Proposition 10 (Immerman). *If K_1 is* **D***–complete for first order reductions, then $FOL[K_1]$ captures* **D** *in the following way: Let $K_0 \in \mathbf{D}$ and let Φ_1 be the first order reduction from K_0 to K_1. Then the formula $Q_{K_1}\Phi_1$ defines K_0.*

However, if we replace K_1 by some K_2 which is also **D**–complete for first order reductions, the formula, which defines K_0 is *not* the formula $Q_{K_2}\Phi_1$. To define K_0, now, we need a first order reduction Φ_2 from K_0 to K_2 and the formula $Q_{K_2}\Phi_2$ then defines K_0.

From this point of view a logic $\mathcal{L}$ captures a complexity class **D** uniformly if it can simulate the Oracle Turing Machines over **D**. The oracle Turing Machines are here viewed as machines pasting together oracle calls (or library programs) without reference to their contents. So the logic needs a certain built in mechanism to simulate **D** modularly.

Our main result in this paper is a negative result. It involves the relativized classes $\mathbf{NC}_i{}^K$ introduced in [Wil86].

Theorem 11. *Let $K_1, \ldots, K_m$ be fixed oracles (languages). There is an oracle language $L(K)$ such that*

(i) $L(K)$ is uniformly recognizable by a uniform family of relativized (oracle) circuits with size $O(n^i)$, for some $i \in \mathbb{N}$ and depth $O((logn)^2)$.
In other words $L(K) \in \mathbf{NC}_2{}^K$ uniformly, and hence $L(K) \in \mathbf{P}^K$ uniformly.

(ii) $L(K)$ is not uniformly definable by any formula $\phi \in FOL[K_1, \ldots, K_m, K]$.

3 Interpretation

3.1 First Order vs Fragments of Second Order Logic

Theorems 5 (7) and 11 spell out a difference over finite structures between First Order Logic with generalized quantifiers and Fixed Point Logic with generalized quantifiers. The former is only capable to capture uniformly complexity classes which use logarithmic space and have a bounded oracle stack. In the case of non–deterministic and alternating logarithmic space additional attention has to be given to the exact oracle computation model. In contrast, the Fixed Point Logics $IFPL, PFPL$ capture uniformly polynomial time (in the Ladner–Lynch model) and space (in the Simon model), respectively.

Although from a model theoretic point of view, a logic whose expressive power lies between First Order Logic and Second Order Logic cannot be classified more or less first or second order. However, from a computational point of view our analysis of generalized quantifiers via the notion of uniformly capturing of complexity classes allows such a distinction. $FOL[ATC]$ has more the character of an extension of First Order Logic, whereas $IFPL$ is more a fragment of Second Order Logic. Both capture $\mathbf{P}$, but $IFPL$ does so uniformly, and therefore reflects better upon the inherent computational power of $\mathbf{P}$.

It is an interesting question, which we do not pursue further in this paper, whether First Order Logic can capture uniformly the oracle classes associated with AC_0.

3.2 Turing Reductions

There is another way of looking at the difference between $FOL[ATC]$ and $IFPL$, which was suggested to us by A. Dawar. On ordered finite structures $FOL[ATC]$ captures $\mathbf{P}$ and also many–one $\mathbf{P}$–reductions (Karp reductions), as shown independently in [Daw93, Daw94] and in [MP93, MP94]. What our result here shows, is that $FOL[ATC]$ does not capture Turing reductions (Cook reductions). The difference is, that in many–one reductions the oracle is applied only once, whereas in Turing reductions it may be applied several times. What $FOL[ATC]$ is missing is the means of propagating the dependency of the next oracle query on the result of a large number of previous queries. In contrast to this $IFPL$ does capture Turing reductions.

This distinction is important, because work in complexity theory has shown that studying reductions allows us to make much finer distinctions. For instance, it is a direct consequence of the Baker, Gill and Solovay result ([BGS75] that there is an oracle K such that $\mathbf{P}^K$ is different from $\mathbf{NP}^K$, that Turing reductions in $\mathbf{P}$ are strictly weaker than Turing reductions in $\mathbf{NP}$. Similarly, in this paper we show that while $FOL[ATC]$ and $ILFP$ both capture computations in $\mathbf{P}$, only the latter captures reductions in $\mathbf{P}$. This separation of the two logics is analogous to the separation of complexity classes in a relativized world.

3.3 A correction

Theorem 11 contradicts a claim made in [MP94]. The reason behind this is, that we overlooked an additional feature of Buss' oracle model for alternating logarithmic space: In his model of relativized $\mathbf{AL}$ it is possible for *different* branches of the computation tree to write information on the *same* oracle tape *in parallel*, a feature which FOL cannot accommodate. The model we defined omits this mechanism. For Buss' model we still have that $\mathbf{AL}^{bK} = \mathbf{AL}^{uK} = \mathbf{P}^K$ for every K. In the model $\mathbf{AL}^{u(\infty)K}$ we use, there are K_0 and a language $L(K_0)$ such that

(i) $L(K_0) \in \mathbf{NC_2}^{K_0} \subseteq \mathbf{P}^{K_0}$ but
(ii) $L(K_0)$ is not definable uniformly in $FOL[ATC, K_0]$.

Now, as $FOL[ATC, K]$ captures $\mathbf{AL}^{u(\infty)K}$ uniformly, we get additionally that

Proposition 12. *There exists an oracle K, such that* $\mathbf{AL}^{u(\infty)K} \neq \mathbf{AL}^{bK}$, *and hence* $\mathbf{AL}^{u(\infty)K} \neq \mathbf{P}^{K}$.

For the oracle computation models of Ladner and Lynch ([LL76], Simon ([Sim77]) and Ruzzo, Simon and Tompa ([RST84]) it was known that oracles K exist with $\mathbf{AL}^{K} \neq \mathbf{P}^{K}$. As a matter of fact, J. Buss introduced his oracle consultation model to fix these anomalies. The oracle we use in the proof of theorem 11 is a slight modification of the oracle used in [Bus88].

4 Standard Model Checkers (SMC)

4.1 Model Checking

Let $K_1 \ldots K_m$ be a set of sets of structures and let $Q_i \ldots Q_m$ be their respective Lindström quantifiers. Let $\mathcal{E} = FOL[Q_1 \ldots Q_m]$, $\mathcal{F} = FPL[Q_1 \ldots Q_m]$ and $\mathcal{S} = SOL[Q_1 \ldots Q_m]$. For $\mathcal{L} = \mathcal{E}, \mathcal{F}, \mathcal{S}$ and their sublogics, we define inductively standard model checkers for all formulas $\phi \in \mathcal{L}$ as follows:

Definition 13 (Assignments). For formulas with free variables the input of a standard model checker is a structure and an *assignment* for the free variables.

(i) For first order variables, the assignment $z(x)$ gives as a value an element of the structure.
(ii) For second order variables of arity k $z(U)$ gives as a value a subset of the k-fold cartesian product of the structure.
(iii) For formulas without free variables the input of a standard model checker is just a structure.

The output is always a yes or a no.

Definition 14 (SMC). (i) If ϕ is T (F) the $SMC(\phi)$ simply returns yes (no).
(ii) If ϕ is of the form $R(x_1 \ldots x_n)$ then the $SMC(\phi)$ checks if $\mathcal{A}, \mathcal{Z} \models \phi$ in some standard way.
(iii) If ϕ is of the form $\psi_1 \wedge \psi_2$, ($\psi_1 \vee \psi_2$, $\neg\psi$) $SMC(\phi)$ calls the model checkers of ψ_1 and ψ_2 and accepts if both accept (if one of them accepts, if $SMC(\psi)$ rejects).
(iv) If ϕ is of the form $\exists x \psi(x)$ ($\forall x \psi(x)$) then for each possible substitution of x the model checker of $\psi(x)$ is invoked with z modified by this substitution for x. $SMC(\phi)$ accepts if one (all) these model checkings accept.
Note that if the structure has cardinality n, then this part may be executed n-times, but these checks do not depend on each other.
(v) If ϕ is of the form $\exists U \psi(U)$ ($\forall U \psi(U)$) then for each possible substitution of U the model checker of $\psi(U)$ is invoked with z modified by this substitution for U. $SMC(\phi)$ accepts if one (all) these model checkings accept.
Note that if the structure has cardinality n, then this part may be executed 2^{kn}-times, but these checks do not depend on each other.

(vi) If ϕ is of the form $\mu U \psi(U)$ then the model checker computes the inflationary fixed point S_d of $\psi(U)$ and then checks $\psi(U)$ for $z(U) = S_d$.
Note that if the structure has cardinality n, then $d = d(n)$ is a polynomial in n. However, here each iteration depends on the outcome of the previous stage.

(vii) If ϕ is of the form $Q_i \Psi$ then $SMC(\phi)$ has one oracle tape using K_i for each occurrence of Q_i. On it it writes the structure $\Psi(\mathcal{A})$ and answers yes if the oracle accepts and no otherwise.
Recall that $\Psi(\mathcal{A})$ is the structure obtained from $\mathcal{A}$ by interpreting the formulas of Ψ in $\mathcal{A}$. To obtain $\mathcal{A}$ we need to invoke the SMCs of the various subformulas of Ψ.

Theorem 15.

(i) Let $\phi \in \mathcal{E}$. Then there is a OTM M_ϕ using logarithmic space such that for every choice of oracles $K_1, \ldots, K_m$, the machine M_ϕ is a SMC for ϕ where the quantifiers are interpreted by $K_1, \ldots, K_m$.

(ii) Let $\phi \in \mathcal{F}$. Then there is a polynomial time OTM M_ϕ such that for every choice of oracles $K_1, \ldots, K_m$ M_ϕ is a SMC for ϕ where the quantifiers are interpreted by $K_1, \ldots, K_m$.
A fortiori, this is also true for $\phi \in \mathcal{E}$.

(iii) Let $\phi \in \mathcal{S}$. Then there is a exponential time OTM M_ϕ such that for every choice of oracles $K_1, \ldots, K_m$ M_ϕ is a SMC for ϕ where the quantifiers are interpreted by $K_1, \ldots, K_m$.

(iv) Actually, for $\phi \in \mathcal{S}$ the OTM uses polynomial space and its time limitations are within the polynomial hierarchy **PH**.

Proof. Easy, from the description of the standard model checker.

4.2 Oracle Interdependence

Definition 16 (Q–rank). For formulas of $\mathcal{E}, \mathcal{F}$ and $\mathcal{S}$ we define the Q–rank as the number of nestings of the generalized quantifiers.

The next lemma spells out the crucial property which makes $\mathcal{E} = FOL[Q_1 \ldots Q_m]$ different from $\mathcal{F} = FPL[Q_1 \ldots Q_m]$. It has to do with the interdependence of oracle consultations. In the case of $\mathcal{E}$ we can layer all possible oracle consultations into finitely many layers, say d_ϕ, a number which depends only on ϕ. Already in $\mathcal{F}$, the number of layers will depend on the structure $\mathcal{A}$. Let us make this precise.

Definition 17. Let $\mathcal{A}$ be a τ–structure and $\phi \in \mathcal{E}(\tau)$. We define inductively sets of structures parametrically definable in $\mathcal{A}$ as follows:

(i) For every subformula $\psi = Q\bar{x}\Phi(\bar{x}, \bar{y})$ of ϕ let $I_{\phi,\psi}(\mathcal{A})$ consist of all the structures $\mathcal{B} = \Phi(\bar{x}, \bar{a})(\mathcal{A})$. Here, $\Phi(\bar{x}, \bar{a})(\mathcal{A})$ denotes the structure defined by Φ in $\mathcal{A}$.
Note that in the case of SOL $\bar{y}$ and $\bar{a}$ can contain second order variables and relation symbols, respectively.

(ii) $I_{\phi,j}(\mathcal{A})$ consists of the union of all the $I_{\phi,\psi}(\mathcal{A})$, where ψ is a subformula of ϕ of Q–rank $j+1$.

(iii) $I_{\phi}(\mathcal{A})$ consists of the union of all the $I_{\phi,\psi}(\mathcal{A})$, where ψ is any subformula of ϕ.

Lemma 18 (Query Dependency I).

(i) For every formula $\phi \in FOL[Q_1,\dots,Q_m]$ there is a polynomial P (in the size of $\mathcal{A}$) such that for every structure $\mathcal{A}$ of size n, I_ϕ contains at most $P(n)$ elements.

(ii) For every formula $\phi \in SOL[Q_1,\dots,Q_m]$ there is a polynomial P (in the size of $\mathcal{A}$) such that for every structure $\mathcal{A}$ of size n, I_ϕ contains at most $2^{P(n)}$ elements.

(iii) Let $\phi \in SOL[Q_1,\dots,Q_m]$ and ψ be a subformula of ϕ of Q–rank $j+1$. Then the evaluation of $SMC(\phi)$ at stage $SMC(\psi)$ depends only on $I_{\phi,j}(\mathcal{A})$.
In other words, the interdependency of oracle consultations is of depth bounded by the Q–rank d_ϕ of ϕ.

Proof. Easy, from the definitions.

Remark. For $\phi \in MFPL[Q_1,\dots,Q_m]$ the lemma is not true. In the evaluation of the fixed point, intermediate relations have to be computed which do depend on each other.

Lemma 19 (Query Dependency II). *Let $\phi \in FOL[Q_1,\dots,Q_m]$ be given. Then there is a constant k_ϕ and a polynomial $P(n) \leq n^{k_\phi}$ such that for every choice of oracles $K_1,\dots,K_m$ and for every structure $\mathcal{A}$ and every assignment z $SMC(\phi,\mathcal{A},z)$ asks at most $P(n)$ many queries.*

Proof. Easy, from the definitions and the previous lemma.

We use this lemma to justify the following

Definition 20 (Diagonalization parameter). We denote, for ϕ as above, by $\nu(\phi)$ the smallest number n such that

$$n^{k_\phi} \leq 2^n.$$

$\nu(\phi)$ is called the *diagonalization parameter of ϕ.*

5 The Language $L_0(K)$

Here we consider binary languages (subsets of $\{0,1\}^*$) and use them both as oracles and as sets of structures. We use elements of $\{0,1\}^*$ also to denote natural numbers in binary notation. For $x \in \{0,1\}^*$ we denote by $l(x)$ the length of x.

Given an oracle K, we define our language $L_0(K)$, which follows an idea of J. Buss [Bus88], section 4.1. He uses his language to separate $\mathbf{NL}^K$ from $\mathbf{P}^K$. Our definitions below are basically a special case of his.

First we associate with an oracle K (a set of strings) an infinite string $S(K)$:

Definition 21. For an arbitrary oracle K, let $S(K)$ be the following infinite binary string: s_1 - the first bit of $S(K)$ is 1 if "0" $\in K$ and 0 otherwise. s_i - the i'th bit - is 1 if $s_1 \dots s_{i-1} \in K$ and 0 otherwise.

Given an infinite string S we can associate with it two languages $K_+(S)$ and $K_-(S)$:

Definition 22. For an arbitrary oracle infinite string S we define

$$K_+(S) = \{s \in \{0,1\}^* : s1 \text{ is an initial segment of } S\}$$

and

$$K_-(S) = \{s \in \{0,1\}^* : s0 \text{ is an initial segment of } S\}.$$

Example 1. Let $K_1 = \{1^n0^n : n \in \mathbb{N}\}$. Then $S(K_1)$ is the infinite string consisting of 0's only.
$K_+(S(K_1)) = \emptyset$ and $K_-(S(K_1)) = \{0\}^+$.
The same is true if we replace K_1 by $K_2 = \{0^n1^n : n \in \mathbb{N}\}$.

Example 2. Let $K_3 = \{01\}^*0$. Then $S(K_3)$ is the infinite string $1000000000\dots$.
$K_+(S(K_3)) = \{\epsilon\}$, i.e. it consists of the empty word alone, and $K_-(S(K_3)) = \{1(0^*)\}$.

More generally, we have the following

Observation 23. *Let K_+ and K_- be two disjoint languages in $\{0,1\}^*$ and $K_i, i = 1,2$ be two languages with $K_+ \subseteq K_i \subseteq \bar{K}_-$. Then*

(i) $S(K_1) = S(K_2)$ and
(ii) for $S = S(K_1) = S(K_2)$, $K_+(S) = K_+$ and $K_-(S) = K_-$.

Next we define, given an infinite string S, languages $L(S)$, $L_1(S)$ and $L_2(S)$ as follows:

Notation 24. *For any number x we denote by $loglog(x)$ the function $log(log(x))$ and by $logloglog(x)$ the function $log(log(log(x)))$*

Definition 25. Let x be a binary word. Let $b_1(x) = \lfloor log^2(l(x)) \rfloor$ and $b_2(x) = \lfloor log(l(x))loglog(l(x)) \rfloor$, where $l(x)$ denotes the length of x.
For an arbitrary infinite string S, the languages $L(S)$ ($L_1(S)$, $L_2(S)$) are defined as follows: $x \in L(S)$ iff the $l(x)$'th ($b_1(x)$'th $b_2(x)$'th) bit of the sequence S is 1. If $S = S(K)$ we write $L_0(K)$ instead of $L(S(K))$ and similarly for L_1 and L_2.

We note the following

Observation 26.

(i) Let S_1 and S_2 be two infinite strings. $L(S_1) = L(S_2)$ iff $S_1 = S_2$.
(ii) If S is an infinite string and x, y are two words and $l(x) = l(y)$ then $x \in L(S)$ iff $y \in L(S)$.
The same is true for L_1 and L_2.
(iii) If K is recognizable by a polynomial TM then so is $L_0(K)$, but the machine depends on K.

6 Capturing $L_0(K)$ Uniformly

We now discuss the complexity of $L_0(K)$. We do this in stages, to make the idea more transparent.

Lemma 27. *There is a OTM M which runs in polynomial time, such that $M(K)$ recognizes $L_0(K)$ uniformly.*

Proof. M keeps one tape T reserved for the bits of $S(K)$. In the first step it writes 0 on the oracle tape. It then iterates the following procedure: query the oracle, if it accepts add a 1 to T else add a 0, then copy the contents of T to the oracle tape. In each iteration the head over the input tape is moved one position to the left, when all the input is scanned accept if the leftmost bit of T is a 1.

We can improve upon this lemma, using relativized uniform circuits, as introduced in [Wil86].

Proposition 28. *There is an* **L***–uniform family of oracle boolean circuits (OBC) C in relativized* $\mathbf{NC}_3$ *such that when the oracle gates of C recognize strings in K Then C recognizes $L_1(K)$.*

Proof. For inputs of size n let $m = \lfloor log^2(n) \rfloor$. The circuit c_n is built from exactly m oracle gates connected as follows: gate 1 has a single input 0. Every gate $i > 1$ has $i - 1$ inputs which are the outputs of all the previous gates. The output of the m'th gate is the output of the circuit.

Clearly, given m, this circuit can be constructed in **L**. Furthermore, m is **L**–computable from the input, so this family of circuits is **L**–uniform.

Also this circuite has a sublinear number of gates, and is of a depth $O(log^2(n))$ oracle gates. As each oracle gate is of depth $log(k)$ where k is the size of the input to that gate, this gate is not in relativized NC_2, but is in relativized NC_3 (it depth is less than $O(log^3(n))$).

Also, given that the oracle gates recognize a set K, this family of circuits recognizes $L_1(K)$ (the output for each gate i is the i'th bit of $S(K)$).

Using the same idea, but working harder, we get

Theorem 29. *There is an* **L***–uniform family of boolean circuits C in relativized* $\mathbf{NC}_2$ *such that when the oracle gates of C recognize strings in K Then C recognizes $L_2(K)$.*

Proof. For inputs of size n let $m = \lfloor log(n)loglog(n) \rfloor$. The circuit c_n is built from exactly m oracle gates connected as follows: gate 1 has a single input 0. Every gate $i > 1$ has $i - 1$ inputs which are the outputs of all the previous gates. The output of the m'th gate is the output of the circuit.

Given that the oracle gates recognize a set K, this family of circuits clearly recognizes $L_2(K)$ (the output for each gate i is the i'th bit of $S(K)$).

Clearly given m this circuite can be constructed in **L** but m is **L**–computable from the input, so this family of circuits is **L**–uniform. Also this circuite has a sublinear number of gates.

Claim 30. *The depth of these circuits is $O(log^2(n))$.*

Clearly the depth of the circuit is exactly m oracle gates. As each oracle gate is of depth $log(k)$ where k is the size of the input to that gate, the depth of the circuite is given by

$$O[1+\sum_{i=2}^{m} log(i-1)] = O[\sum_{i=1}^{m} log(i)]$$

(The first gate has one input and all other $i-1$ inputs). However

$$\sum_{i=1}^{m} log(i) = log(m!) \approx log(\sqrt{2\pi m}m^m e^{-m})$$

Where the approximation is via Stirling's formula.

$$log(\sqrt{2\pi m}m^m e^{-m}) = O(log(m)) + mlog(m) - O(m) = O(mlog(m))$$

Substituting for m we get

$$\begin{gathered} mlog(m) = (log(n)loglog(n)) \cdot log(log(n)loglog(n)) = \\ (log(n)loglog(n)) \cdot (loglog(n) + logloglog(n)) = \\ O(log(n)loglog^2(n)) < O(log^2(n)) \end{gathered}$$

7 The Diagonalization

We are now ready for our main result.

Theorem 31.

(i) For no set of oracles $K_1, \ldots, K_m$ does $\mathcal{E} = FOL[K_1, \ldots, K_m, K]$ capture $\mathbf{P}^K$ uniformly.
(ii) In particular, $\mathbf{P}^K$ cannot be uniformly captured by formulas of $FOL[ATC, K]$.

Proof. For an arbitrary oracle K, let $L_0(K)$ be as in definition 25 and let $M(K)$ be the OTM from lemma 27 which recognizes $L_0(K)$ uniformly. To prove the theorem it is sufficient to show that no $\phi(K) \in \mathcal{E}$ defines $L_0(K)$ uniformly. Or in other words:

Given a $\phi \in \mathcal{E}$, there is a structure $\mathcal{A}$ and two sets of structures K and K' such that $\mathcal{A} \in L(M,K)$ and $\mathcal{A} \notin L(M,K')$ but $\mathcal{A} \in Mod(\phi, K)$ and $\mathcal{A} \in Mod(\phi, K')$.

Assume, for contradiction, that $\phi(K) \in \mathcal{E}(\tau)$ is τ–sentence which defines $L_0(K)$ uniformly. Let $\mathcal{A}$ be a τ–structure of size $n \geq \nu(\phi)$, where $\nu(\phi)$ is from definition 20.

We are going to construct two oracles K_0, K_1, depending on $\mathcal{A}$, such that $\mathcal{A} \notin L_0(K_0)$ $\mathcal{A} \in L_0(K_1)$. The oracles will differ exactly on the string w which is used by $M(K)$ in an oracle consultation, i.e. on which the language $L_0(K)$

depends, but which, on the other hand, is not used in the evaluation of $\mathcal{A} \models \phi(K)$.

In the definition of $L_0(K)$ we use the infinite string $S(K)$, cf. definition 21. Let $S(K)_j$ denote the first j bits of $S(K)$. The string w on which K_0 and K_1 will differ is going to be $S(K)_{n-1}$.

To make all this precise we prove two lemmas.

The first lemma is obvious, once the right definitions are given.

Lemma 32 (Dependency Lemma). *Let K_1 be an oracle, $w \in K_1$ and $K_2 = K_1 - \{w\}$. Let $\phi \in \mathcal{E}$ and $\mathcal{A}$ be a structure. If $w \notin I_\phi(\mathcal{A})$ then*

$$\mathcal{A} \in Mod(\phi, K_1) \textit{ iff } \mathcal{A} \in Mod(\phi, K_2).$$

The second lemma is the core of the diagonalization: It is proved by induction on the quantifier depth.

Lemma 33. *For every $\phi \in \mathcal{E}$ there is an $n_0 \in \mathbb{N}$ such that for every $\mathcal{A}$ of size n bigger than n_0 there is an oracle K_A with $S(K_A)_{n-1} \notin I_\phi(\mathcal{A})$.*
In fact, $n_0 = \nu(\phi)$, the diagonalization parameter from definition 20, suffices.

Proof. Let n denote the size of $\mathcal{A}$ and k_ϕ be from lemma 19. We prove the lemma by induction on i the index of the $I_{\phi,i}$'s and use the fact that there are only d $=Q$–rank of ϕ many of them. For each $I_{\phi,i}$ we modify our oracle K (and thus change $k_\phi \times log(n)$ bits of $S(K)$) such that all queries in $I_{\phi,i}$ are either shorter than $i \times k_\phi \times log(n)$ or at least one of their first $(i \times k_\phi \times log(n)) + 1$ bits is different from the bit in the same place in $S(K)_{n-1}$.

We now proceed by induction:

Basis: $I_{\phi,1}$ queries depend only on $\mathcal{A}$. We choose the first $k_\phi \times log(n)$ bits of $S(K)$ as follows: If the majority of queries in $I_{\phi,1}$ start with a bit 1 we chose the first bit of $S(K)$ to be 0 and vice versa. Clearly, after this choice, already half of the queires in $I_{\phi,1}$ are in the desired form. For the j'th bit we consider only those queries of $I_{\phi,1}$ which match $S(K)$ on the first $j-1$ bits. Among those if the majority starts with bit x we choose bit $\bar{x}$ (x XOR $\bar{x} = 1$).

Clearly, after $k_\phi \times log(n)$ bits are chosen, all queries of $I_{\phi,1}$ are either shorter than $i \times k_\phi \times log(n)$ or at least one of their first $(i \times k_\phi \times log(n)) + 1$ bits is different from the bit in the same place in $S(K)_{n-1}$ (which is as yet determined only up to its first $k_\phi \times log(n)$ bits).

Note that, in this procedure, we have determined the answers only for those queries which were possible prefixes of $S(K)$ as constructed so far. For the other queries in $I_{\phi,1}$ we set the answers arbitrarily. We must fix these answers too, if the queries in $I_{\phi,2}$ are to be uniquely determined.

Induction step: Assume we have succeeded in modifying the queries and answers of $I_{\phi,1} - I_{\phi,i-1}$ and hence, we have modified the first $(i-1) \times k_\phi \times log(n)$ bits of $S(K)$. Then we can modify $I_{\phi,i}$ as follows:

First we look at all queries of $I_{\phi,i}$ which already appeared at previous levels. To these we set the answers they had in the previous levels (note that they are already either shorter than $(i-1) \times k_\phi \times log(n)$ or at least one of their first $((i-1) \times k_\phi log(n))+1$ bits is different from the bit in the same place in $S(K)_{n-1}$ as they match a query from a previous level).

For new queries which are already in the desired form we set arbitrary answers.

For the remaining queries we ensure our condition as we did for $I_{\phi,1}$ by iterating the following procedure: In each iteration we consider only queries such that their first $((i-1) \times k_\phi \times log(n))+j$ bits coincide with the determined prefix of $S(K)$. If the majority of these have x as their $((i-1) \times k_\phi \times log(n))+j+1$'th bit we choose $\bar{x}$ as the next bit of $S(K)$, as in the basis of the induction.

Clearly, after $k_\phi \times log(n)$ such iterations all queries of $I_{\phi,i}$ are in the desired form. As before for those queries whose answers are not determined during this process we set arbitrary answers.

After we have completed the above procedure for all $I_{\phi,i}$'s up to I_{ϕ,d_ϕ}, we have determined the answers of an oracle K_A to all queries asked for checking $\mathcal{A} \models \phi$ and hence we have determined the answer of the model checker for $\mathcal{A} \models \phi$ for oracle K_A. However, in fact we have determined the answer of $\mathcal{A} \models \phi$ for any oracle K which gives the same answers as K_A to the queries in $I_\phi = \bigcup_{1 \leq i \leq d_\phi} I_{\phi,i}$

However, none of this polynomial set of queries is $S(K)_{n-1}$. Hence, using lemma 32, we have the freedom to select K and K' such that they coincide on all asked queries, but differ on $S(K)_{n-1}$. Therefore either $\mathcal{A} \models \phi$ or $\mathcal{A} \not\models \phi$ for both K and K'. But $\mathcal{A} \in L_0(K)$ iff $\mathcal{A} \notin L_0(K')$ and thus we prove ϕ does not uniformly capture M_K.

Using the languages $L_1(K)$ or $L_2(K)$ from proposition 28 and theorem 29 respectively, we can actually prove:

Theorem 34.

(i) Let $K_1, \ldots, K_m$ be a fixed set of oracles. Let $\mathcal{E} = FOL[K_1, \ldots, K_m, K]$ be a finitely generated Lindström Logic $L_1(K)$ ($L_2(K)$) be the oracle languages from section 5. Then no $\phi \in \mathcal{E}(\tau_{diag})$ defines $L_1(K)$ ($L_2(K)$) uniformly.

(ii) In particular, there are oracle languages $L_0(K)$ in $\mathbf{NC_3}^K$ ($\mathbf{NC_2}^K$) which are not uniformly definable in any finitely generated Lindström Logic.

8 Conclusions and Conjectures

We have introduced the notion of Logics capturing relativized (= oracle) complexity classes uniformly via generalized quantifiers. We have shown that First Order Logic with generalized quantifiers can only capture comlexity classes related to Logspace, and even then restrictions on the oracle computation model have to made.

On the other hand, Fixed Point Logics and Second Order Logic are capable of capturing quite a wide class of relativized polynomial time complexity classes uniformly. Our main result states, that First Order Logic cannot do this for classes containing relativized $\mathbf{NC_2}$.

It seems natural to ask, whether First Order Logic is not more appropriate for capturing relativized classes contained in $\mathbf{NC}_2$.

We end this paper with a problem.

Problem 35. In what sense does $FOL[K]$ capture relativized ${\mathbf{AC}_0}^K$?

We would expect a positive answer, but possibly with a proviso. It would fit Immerman's results nicely, cf. [Imm89], and complement our results in this paper. However, these low complexity classes are very sensitive to minor variations in the computational model, so we are prepared for surprises.

References

[AHV94] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Database.* Addison Wesley, 1994.

[BF85] J. Barwise and S. Feferman, editors. *Model-Theoretic Logics.* Perspectives in Mathematical Logic. Springer Verlag, 1985.

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the P =? NP question. *SIAM Journal for Computing*, 4:431–442, 1975.

[Bus86] J. Buss. Relativized alternation. In *Structure in Complexity Theory*, volume 223 of *Lecture Notes in Computer Science*, pages 66–103. Springer Verlag, 1986.

[Bus88] J.F. Buss. Alternations and space-bounded computations. *Journal of Computer and System Sciences*, 36:351–378, 1988.

[Daw93] A. Dawar. *Feasible Computation Through Model Theory.* PhD thesis, Department of Computer Science, University of Maryland, 1993.

[Daw94] A. Dawar. Generalized quantifiers and logical reducibilities. *Logic and Computation*, to appear, 1995.

[EF95] H.D. Ebbinghaus and J. Flum. *Finite Model Theory.* In preparation, 1995.

[EFT80] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic.* Undergraduate Texts in Mathematics. Springer-Verlag, 1980.

[GJ79] M.G. Garey and D.S. Johnson. *Computers and Intractability.* Mathematical Series. W.H. Freeman and Company, 1979.

[Got95a] G. Gottlob. Relativized logspace and generalized quantifiers over finite structures. TR CD-TR-95/76, Technical University of Vienna, 1995.

[Got95b] G. Gottlob. Relativized logspace and generalized quantifiers over finite structures. In *LiCS'95*, to appear. IEEE, 1995.

[GS86] Y. Gurevich and S. Shelah. Fixed point extensions of first order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, Aug 1987.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[Imm89] N. Immerman. Expressibility and parallel complexity. *SIAM Journal on Computing*, 18:625–638, 1989.

[Lei90] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95–108, 1990.

[LL76] R.E. Ladner and N. Lynch. Relativization of questions about log-space reducibility. *Mathematical Systems Theory*, 10:19–32, 1976.

[MP93] J.A. Makowsky and Y.B. Pnueli. Computable quantifiers and logics over finite structures. To appear in 'Quantifiers: Generalizations, extensions and and variants of elementary logic', Kluwer Academic Publishers, preliminary version TR 768, Department of Computer Science, Technion–Israel Institute of Technology, Haifa, Israel, 1993.

[MP94] J.A. Makowsky and Y.B. Pnueli. Oracles and quantifiers. In *CSL'93*, volume 832 of *Lecture Notes in Computer Science*, pages 189–222. Springer, 1994.

[MP95] J.A. Makowsky and Y. Pnueli. Second order logics capturing complexity classes. in preparation, 1995.

[RST84] W.L. Ruzzo, J. Simon, and M. Tompa. Space bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28:216–230, 1984.

[Sim77] I. Simon. *On some subrecursive reducibilities.* PhD thesis, Department of Computer Science, Stanford University, 1977.

[Ste94] I. Stewart. Incorporating generalized quantifiers and the least fixed point operator. In *CSL'93*, volume 832 of *Lecture Notes in Computer Science*, pages 318–333. Springer, 1994.

[Var82] M. Vardi. The complexity of relational query languages. In *STOC'82*, pages 137–146. ACM, 1982.

[Wil86] C.B. Wilson. Parallel computation and the NC hierarchy relativized. In *Structure in Complexity Theory*, volume 223 of *Lecture Notes in Computer Science*, pages 362–382. Springer Verlag, 1986.

Preservation Theorems in Finite Model Theory*

Eric Rosen** and Scott Weinstein***

Department of Philosophy
University of Pennsylvania
Philadelphia PA 19104, USA

Abstract. We develop various aspects of the finite model theory of $L^k(\exists)$ and $L^k_{\infty\omega}(\exists)$. We establish the optimality of normal forms for $L^k_{\infty\omega}(\exists)$ over the class of finite structures and demonstrate separations among descriptive complexity classes within $L^k_{\infty\omega}(\exists)$. We establish negative results concerning preservation theorems for $L^k(\exists)$ and $L^k_{\infty\omega}(\exists)$. We introduce a generalized notion of preservation theorem and establish some positive results concerning "generalized preservation theorems" for first-order definable classes of finite structures which are closed under extensions.

1 Introduction

In this paper we investigate the status of preservation theorems in finite model theory. We focus our attention on classes of finite structures which are closed under extensions and their definability in fragments of the infinitary language $L^\omega_{\infty\omega}$. The language $L^\omega_{\infty\omega}$was introduced by Barwise [4] in connection with the investigation of inductive definability over infinite structures. Recently, the study of $L^\omega_{\infty\omega}$ has played a central role in analyzing the behavior of fixed-point logics over the class of finite structures (see [5, 13]). Of particular interest from the point of view of our current investigation are the works of Kolaitis and Vardi [12] and Afrati, Cosmadakis, and Yannakakis [1] which exploit existential fragments of $L^\omega_{\infty\omega}$ in analyzing the expressive power of Datalog.

The starting point for our investigation is the well-known failure of the preservation theorem of Los and Tarski over finite structures. Recall that the Los-Tarski Theorem states that any first-order definable class of structures which is closed under extensions is definable by a first-order existential sentence. Scott and Suppes conjectured that this theorem generalizes to the finite case, that is, if $\mathrm{Mod}_f(\varphi)$ (the collection of finite models of the first-order sentence φ) is closed under extensions, then $\mathrm{Mod}_f(\varphi) = \mathrm{Mod}_f(\psi)$, for some first-order existential sentence ψ. Tait [18] showed that this conjecture fails; Gurevich and Shelah [9, 10] gave simpler counterexamples employing universal-existential first-order sentences.

* We would like to thank Maria Bonet, Yuri Gurevich, and Steven Lindell for valuable discussions on the subject of this paper.

** Supported in part by NSF-STC SBR-8920230.

*** Supported in part by NSF CCR-9403447.

In light of the failure of the Los-Tarski Theorem over finite structures, it is natural to inquire whether "generalized preservation theorems" might hold in the finite case. In this paper, we investigate the prospects for such a positive approach to preservation properties in the context of finite model theory. In particular, we examine generalized versions of ordinary preservation theorems where an algebraic restriction on a class of structures definable in a given language yields information about the syntactic structure of formulas which define that class in an *extension* of that language. In this spirit, we show that for certain classes of first-order sentences Φ, if $\varphi \in \Phi$ and $\mathrm{Mod}_f(\varphi)$ is closed under extensions, then $\mathrm{Mod}_f(\varphi) = \mathrm{Mod}_f(\psi)$ for some ψ in the existential fragment of $L^{\omega}_{\infty\omega}$ (or even in Datalog($\neq, \neg$)). In contrast, we also establish the failure of the analog of the Los-Tarski Theorem for $L^{\omega}_{\infty\omega}$ itself, both over finite structures and over arbitrary structures. That is, we show that there is a sentence φ of $L^{\omega}_{\infty\omega}$ such that both $\mathrm{Mod}_f(\varphi)$ and $\mathrm{Mod}(\varphi)$ are closed under extensions, but neither of these classes is definable by an existential sentence of $L^{\omega}_{\infty\omega}$.

The paper proceeds as follows. The next section introduces the languages we will study and establishes a simple proposition which characterizes the relative expressive power of their existential fragments. Section 3 develops some finite model theory for the existential fragments of L^k and $L^k_{\infty\omega}$. In particular, we establish the optimality of a normal form for the existential fragment of $L^k_{\infty\omega}$ over finite structures and demonstrate separations among descriptive complexity classes within $L^k_{\infty\omega}(\exists)$. In Section 4, we prove the failure of existential preservation for $L^{\omega}_{\infty\omega}$. Section 5 is devoted to establishing positive results concerning generalized preservation theorems for fragments of first-order logic over finite structures. In the final section, we discuss a number of open problems and present without proof some related results concerning preservation under homomorphisms. A full treatment of these results will appear in [17].

2 Preliminaries

Let $\mathcal{F}_\sigma$ be the collection of finite structures of signature σ. We will assume that the universe of any $A \in \mathcal{F}_\sigma$ is an initial segment of $N = \{0, 1, 2, \ldots\}$. We will often use $A, B, \ldots$ etc. to denote both a structure and its universe when no confusion is likely to result. We assume that the signature σ is finite and contains no function symbols; we suppress mention of σ when no confusion is likely to result. A *boolean query* $\mathcal{C} \subseteq \mathcal{F}$ is a class of finite structures that is closed under isomorphisms. We use $\mathcal{C}$ to range over boolean queries. In what follows, we will focus attention on boolean queries which are closed under extensions.

Definition 1. $\mathrm{EXT} = \{\mathcal{C} \subseteq \mathcal{F} \mid \forall A, B \in \mathcal{C},$ if $A \in \mathcal{C}$ and $A \subseteq B$, then $B \in \mathcal{C}\}$.

Let L be a logical language and let φ be a sentence of L. $\mathrm{Mod}(\varphi) = \{A \mid A \models \varphi\}$ is the *L-class* determined by φ and $\mathrm{Mod}_f(\varphi) = \{A \in \mathcal{F} \mid A \models \varphi\}$ is the boolean query expressed by φ. We say that $\mathcal{C}$ is *L-definable*, just in case it is the boolean query expressed by some sentence $\varphi \in L$. We will often use L to denote the set of L-definable boolean queries. We let FO denote first-order logic, $L_{\infty\omega}$, the usual infinitary extension of first-order logic which allows conjunction and disjunction over arbitrary sets of formulas, L^k, the fragment of FO consisting of those formulas all of whose variables both free and bound are among $x_1, \ldots, x_k$, and similarly $L^k_{\infty\omega}$, the k-variable fragment of $L_{\infty\omega}$; $L^\omega_{\infty\omega} = \bigcup_{k \in \omega} L^k_{\infty\omega}$. We let FO($\exists$) denote the set of existential formulas of FO, that is, those formulas obtained by closing the set of atomic formulas and negated atomic formulas under the operations of conjunction, disjunction, and existential quantification. We define $L_{\infty\omega}(\exists)$, the set of existential formulas of $L_{\infty\omega}$, similarly, but require, in addition, closure under infinitary conjunction and disjunction. We let $L^k(\exists)$ consist of the formulas common to FO($\exists$) and L^k and we define $L^k_{\infty\omega}(\exists)$ and $L^\omega_{\infty\omega}(\exists)$ similarly. A Datalog($\neq, \neg$) program P is a collection of rules of the form

$$\eta_0 \longleftarrow \eta_1, \ldots, \eta_k.$$

Such a rule has a *head*, η_0, and a *body*, $\eta_1, \ldots, \eta_k$. Each of the η_i is either an inequality or a literal over the signature $\sigma \cup \tau$ where σ and τ are disjoint; σ consists of the *extensional* relations and constants of P and τ consists of the *intensional* relations of P. The heads of all rules are built from intensional relations and intensional relations occur only positively throughout P. The program contains a distinguished intensional relation R of arity $n \geq 0$ and determines an n-ary query over structures in $\mathcal{F}_\sigma$. The value of this query for a given $A \in \mathcal{F}_\sigma$ is the value of R when the program is viewed as determining least-fixed points for each of the intensional relations with respect to a simultaneous induction associated with the program. The reader may consult [1, 12] for further details and discussion. As with logics, we use Datalog($\neq, \neg$) to refer to the class of queries computed by Datalog($\neq, \neg$) programs as well as to the class of programs themselves. Datalog programs are defined similarly except that all the η_i are restricted to be positive literals, even those built from extensional relations. Observe that Datalog($\neq, \neg$) is contained in the least fixed-point extension of first-order logic (FO+LFP).

In our current notation, the failure of the Los-Tarski Theorem over finite structures may be expressed as:

$$\text{FO} \cap \text{EXT} \not\subseteq \text{FO}(\exists).$$

This raises the question of whether FO $\cap$ EXT is contained in the existential fragment of some stronger logic. The following proposition completely characterizes the relative expressive power of the existential fragments of the logics in which we are interested.

Proposition 2.

$$\text{FO}(\exists) \subset \text{Datalog}(\neq, \neg) \subset L^\omega_{\infty\omega}(\exists) \subset L_{\infty\omega}(\exists) = \text{EXT}.$$

Proof. It is easy to see that every query in FO($\exists$) can be expressed by a program in Datalog($\neq, \neg$) which makes use of no recursion. It is well-known that this inclusion is strict, for example, the query (s, t)-connectivity is expressible in Datalog but not in FO. The inclusion of Datalog($\neq, \neg$) in $L^\omega_{\infty\omega}(\exists)$ has been noted by Afrati, Cosmadakis, and Yannakakis [1] (see also [12]); the argument to show this is a variant of the proof that least fixed-point logic is contained in $L^{\omega}_{\infty\omega}$ over the class of finite structures (see [14]). Afrati, Cosmadakis, and Yannakakis [1] also exhibit queries which witness the separation of Datalog($\neq, \neg$) and $L^{\omega}_{\infty\omega}(\exists)$, even over the class of polynomial time computable queries. The identity between $L_{\infty\omega}(\exists)$ and EXT has been noted by Kolaitis and independently by Lo (see [1] and [15]). Finally, it is easy to construct polynomial time computable boolean queries in EXT which are not in $L^{\omega}_{\infty\omega}$. For example, let $\mathcal{C}$ be the query over the signature $\{E, s, t\}$ of source-target graphs that says that there is an E-path from s to t whose length is less than half the cardinality of the structure. It is clear that $\mathcal{C} \in$ EXT. It is also easy to verify that $\mathcal{C}$ is not in $L^{\omega}_{\infty\omega}$ (and therefore not in $L^{\omega}_{\infty\omega}(\exists)$) by a straightforward application of the k-pebble Ehrenfeucht-Fraisse game which we review below. ■

The above proposition together with the failure of the Los-Tarski Theorem in the finite case suggests the following questions.

1. Is FO $\cap$ EXT $\subseteq L^{\omega}_{\infty\omega}(\exists)$?
2. Is FO $\cap$ EXT $\subseteq$ Datalog($\neq, \neg$)?
3. Is $L^{\omega}_{\infty\omega} \cap$ EXT $\subseteq L^{\omega}_{\infty\omega}(\exists)$?

Clearly a positive answer to the second or third question would imply a positive answer to the first. In Section 4, we provide a negative answer to the third question. In Section 5, we provide partial positive answers to the first and second questions. Before proceeding to these results, we develop some of the finite model theory of $L^k(\exists)$ and $L^k_{\infty\omega}(\exists)$ in the next section.

3 Basic Finite Model Theory for $L^k(\exists)$ and $L^k_{\infty\omega}(\exists)$

In this section, we present some basic model theory for L^k, $L^k_{\infty\omega}$, $L^k(\exists)$, and $L^k_{\infty\omega}(\exists)$. After a brief discussion of game-theoretic characterizations of equivalence and definability in these languages, we proceed to consider questions of finite axiomatizability and normal forms.

Let L be one of the logical languages we consider. Given a structure A, the *L-theory* of A is the collection of sentences of L which are satisfied by A. We say that A is *L-equivalent* to B, if and only if, the L-theory of A is equal to the L-theory of B and we say that A is *L-compatible* with B, if and only if, the L-theory of A is contained in the L-theory of B. Note that if L is closed under negation, then the relations of L-equivalence and L-compatibility coincide, whereas for languages like $L^k(\exists)$ and $L^k_{\infty\omega}(\exists)$ these relations are distinct. We use the notations $\equiv^k$, $\equiv^k_{\infty\omega}$, $\preceq^k$, and $\preceq^k_{\infty\omega}$ for L^k-equivalence, $L^k_{\infty\omega}$-equivalence, $L^k(\exists)$-compatibility, and $L^k_{\infty\omega}(\exists)$-compatibility, respectively. The main tool for studying these relations are refinements of the Ehrenfeucht-Fraisse game. Barwise [4] characterized $L^k_{\infty\omega}$-equivalence in terms of partial isomorphisms, while Immerman [11] and Poizat [16] provided related pebble game characterizations of L^k-equivalence. Kolaitis and Vardi [12] characterized compatibility in the negation free fragment of $L^k_{\infty\omega}(\exists)$ both in terms of collections of partial *homomorphisms* as well as in terms of a one-sided, positive version of the pebble game. Below we use a minor variant of the approach in [12] to characterize $L^k_{\infty\omega}(\exists)$-compatibility.

A collection I of partial isomorphisms from A to B is said to have the k-*[back-and-]forth property* if for all $f \in I$ such that the domain of f has cardinality $< k$, and all $a \in A$ $[b \in B]$, there is a function $g \in I$ such that $f \subseteq g$ and $a \in \mathrm{dom}(g)[b \in \mathrm{rng}(g)]$. (That is, the k-forth property is the one-sided version, going forth from A, of the k-back-and-forth property.)

Barwise [4] proved the following proposition which gives an algebraic characterization of $L^k_{\infty\omega}$-equivalence.

Proposition 3 (Barwise [4]). *Let A and B be structures of signature σ and let h be the map with $\mathrm{dom}(h) = \{c^A \mid c \in \sigma\}$ such that $h(c^A) = c^B$ for all $c \in \sigma$. The following conditions are equivalent.*

1. *$A \equiv^k_{\infty\omega} B$.*
2. *There is a non-empty set I of partial isomorphisms from A to B such that*
   (a) *I is closed under subfunctions;*
   (b) *I has the k-back-and-forth property;*
   (c) *for all $f \in I$, $f \cup h$ is a partial isomorphism from A to B.*

In a similar spirit, Kolaitis and Vardi [12] gave an algebraic characterization of the compatibility relation for the negation free fragment of $L^k_{\infty\omega}(\exists)$ in terms of collections of partial *homomorphisms* with the k-forth property. We adapt their approach to the case of $L^k_{\infty\omega}(\exists)$ in the following theorem.

Proposition 4 (Kolaitis and Vardi [12]). *Let A and B be structures of signature σ and let h be the map with $\mathrm{dom}(h) = \{c^A \mid c \in \sigma\}$ such that $h(c^A) = c^B$ for all $c \in \sigma$. The following conditions are equivalent.*

1. *$A \preceq^k_{\infty\omega} B$.*
2. *There is a non-empty set I of partial isomorphisms from A to B such that*
   (a) *I is closed under subfunctions;*
   (b) *I has the k-forth property;*
   (c) *for all $f \in I$, $f \cup h$ is a partial isomorphism from A to B.*

Both Propositions 3 and 4 can be expressed more colorfully in terms of pebble games. This approach to L^k-equivalence was introduced by Immerman [11] and Poizat [16] and as an approach to $L^k_{\infty\omega}(\exists)$-compatibility by Kolaitis and Vardi [12]. In order to state the relevant results in a suitably refined form, we require the notion of the *quantifier rank* of a formula. We state this definition for formulas of $L_{\infty\omega}$ since all the languages we consider are fragments of it.

Definition 5. The quantifier rank of $\varphi \in L_{\infty\omega}$ ($\mathrm{qr}(\varphi)$) is defined by the following induction.

1. $\mathrm{qr}(\varphi) = 0$ if φ is atomic;
2. $\mathrm{qr}(\neg\varphi) = \mathrm{qr}(\varphi)$;
3. $\mathrm{qr}(\bigwedge \Phi) = \mathrm{qr}(\bigvee \Phi) = \sup(\{\mathrm{qr}(\varphi) \mid \varphi \in \Phi\})$;

4. $\mathrm{qr}(\exists x\varphi) = \mathrm{qr}(\forall x\varphi) = \mathrm{qr}(\varphi) + 1$.

The *n-round, k-pebble Ehrenfeucht-Fraisse game* on A and B is played between two players, Spoiler and Duplicator, with k pairs of pebbles, $(\alpha_1, \beta_1), \ldots, (\alpha_k, \beta_k)$. The Spoiler begins each round by choosing a pair of pebbles (α_i, β_i) that may or may not be in play on the boards A and B. He (by convention, the Spoiler is male, the Duplicator female) either places α_i on an element of A, or β_i on an element of B. The Duplicator then plays the remaining pebble on the other model. The Spoiler wins the game if after any round $m \leq n$ the function f from A to B, which sends the element pebbled by α_i to the element pebbled by β_i and preserves the denotations of constants, is not a partial isomorphism; otherwise, the Duplicator wins the game. The *n-round* $\exists^k$-game is the one-sided version of the n-round, k-pebble Ehrenfeucht-Fraisse game in which the Spoiler is restricted to play a pebble α_i into A at every round while the Duplicator responds by playing β_i into B; the winning condition remains the same. Both the k-pebble Ehrenfeucht-Fraisse game and its one-sided variant have infinite versions, which we call the *eternal* k-pebble Ehrenfeucht-Fraisse game and the *eternal* $\exists^k$-game. In these games, the play continues through a sequence of rounds of order type ω. The Spoiler wins the game, if and only if, he wins at the n^{th}-round for some $n \in \omega$ as above; otherwise, the Duplicator wins. In describing the play of pebble games below, we will often use S to refer to the Spoiler and D to refer to the Duplicator. We will also often use α_i, β_i, etc. to refer to both pebbles and the elements they pebble at a given round of play.

The foregoing n-round games may be used to characterize equivalence and compatibility of structures with respect to L^k sentences and $L^k(\exists)$ sentences of quantifier rank n, and the eternal games may be used to characterize equivalence and compatibility of structures with respect to $L^k_{\infty\omega}$ sentences and $L^k_{\infty\omega}(\exists)$ sentences. Given structures A and B we let $A \equiv^{k,n} B$, if and only if, A and B satisfy the same sentences of L^k of quantifier rank $\leq n$ and we let $A \preceq^{k,n} B$, if and only if, every sentence of $L^k(\exists)$ of quantifier rank $\leq n$, which is true in A, is also true in B. The following two propositions use the n-round pebble games to characterize these relations. The first is due to Immerman [11] and Poizat [16] and the second is essentially due to Kolaitis and Vardi [12].

Proposition 6 (Immerman [11], Poizat [16]). *For all structures A and B, the following conditions are equivalent.*

1. $A \equiv^{k,n} B$.
2. *The Duplicator has a winning strategy for the n-round, k-pebble Ehrenfeucht-Fraisse game on A and B.*

Proposition 7 (Kolaitis and Vardi [12]). *For all structures A and B, the following conditions are equivalent.*

1. $A \preceq^{k,n} B$.
2. *The Duplicator has a winning strategy for the n-round $\exists^k$-game on A and B, with the Duplicator playing on B.*

The next proposition gives a characterization of the infinitary equivalence and compatibility relations in terms of the eternal games. It is essentially due to Kolaitis and Vardi [14, 12].

Proposition 8 (Kolaitis and Vardi [14, 12]). *1. For all structures A and B, the following conditions are equivalent.*

(a) $A \equiv^k_{\infty\omega} B$.

(b) The Duplicator has a winning strategy for the eternal k-pebble Ehrenfeucht-Fraisse game on A and B.

2. For all structures A and B, the following conditions are equivalent.

(a) $A \preceq^k_{\infty\omega} B$.

(b) The Duplicator has a winning strategy for the eternal $\exists^k$-game on A and B, with the Duplicator playing on B.

Kolaitis and Vardi [14, 12] observed that over finite structures infinitary equivalence and compatibility coincide with their finitary analogs.

Proposition 9 (Kolaitis and Vardi [14, 12]). *1. Let A or B be a finite structure. Then, the following conditions are equivalent.*

(a) $A \equiv^k_{\infty\omega} B$.

(b) $A \equiv^k B$.

2. Let B be a finite structure. Then, the following conditions are equivalent.

(a) $A \preceq^k_{\infty\omega} B$.

(b) $A \preceq^k B$.

The foregoing propositions yield the following corollaries concerning definability.

Proposition 10 (Kolaitis and Vardi [12]). *For all $\mathcal{C} \subseteq \mathcal{F}$, the following conditions are equivalent.*

1. *$\mathcal{C}$ is $L^k_{\infty\omega}(\exists)$-definable.*
2. *For all $A \in \mathcal{C}$ and $B \notin \mathcal{C}$, $A \not\preceq^k_{\infty\omega} B$.*
3. *For all $A \in \mathcal{C}$ and $B \notin \mathcal{C}$, $A \not\preceq^k B$.*
4. *For all $A \in \mathcal{C}$ and $B \notin \mathcal{C}$, there is an $n \in \omega$ such that the Spoiler has a winning strategy for the n-round $\exists^k$-game on A and B with the Spoiler playing on A.*

Let L and L' be logical languages and let T be a collection of sentences of L. We say that T is *finitely axiomatizable in L'*, if and only if, there is a sentence $\varphi \in L'$ such that $\mathrm{Mod}_f(T) = \mathrm{Mod}_f(\varphi)$. Dawar, Lindell and Weinstein [5] prove that the $L^k_{\infty\omega}$-theory of any finite model is finitely axiomatizable in L^k. As a corollary, they obtain a simple normal form for $L^k_{\infty\omega}$ over $\mathcal{F}$, in particular, they show that every sentence of $L^k_{\infty\omega}$ is equivalent to a countable disjunction of sentences of L^k and is also equivalent to a countable conjunction of sentences of L^k. In contrast, we show below that there are finite models whose $L^k(\exists)$-theories are not finitely axiomatizable in $L^k(\exists)$. Building on this result, we prove that the normal form for $L^k_{\infty\omega}$ over $\mathcal{F}$ (every sentence of $L^k_{\infty\omega}$ is equivalent over $\mathcal{F}$ to a countable disjunction of countable conjunctions of sentences of L^k) exhibited by Kolaitis and Vardi [14] is optimal when considered as a normal form for $L^k_{\infty\omega}(\exists)$ sentences over $L^k(\exists)$.

We begin by proving that there are models whose $L^k(\exists)$-theories are not finitely axiomatizable in $L^k(\exists)$. Our argument exploits the *k-extension axioms*, which we now describe briefly. Let σ be a purely relational, finite signature. A *basic k-type* π over the signature σ is a maximal consistent set of literals over σ in the variables $x_1, \ldots, x_k$. A *k-extension axiom* of signature σ is a sentence of the form $\forall x_1 \ldots x_{k-1} \exists x_k (\bigwedge \pi \rightarrow \bigwedge \pi')$, where π is a basic $(k-1)$-type of signature σ, π' is a basic k-type of signature σ, and $\pi \subseteq \pi'$. Over a fixed signature σ, the *k-Gaifman theory*, Γ_k, is the set of all k-extensions axioms of signature σ. It is easy to see that, for each k, there are only finitely many k-extension axioms. Gaifman [7] showed that the theory $T = \bigcup_k \Gamma_k$ axiomatizes an ω-categorical model called the *random structure*. Fagin [6] proved the 0-1 law for first-order logic by showing that every extension axiom is *almost surely true* over $\mathcal{F}$. Fagin's result implies that almost every $A \in \mathcal{F}$ satisfies the k-Gaifman theory. Immerman [11] showed that any two models of the k-Gaifman theory are L^k-equivalent and Kolaitis and Vardi [14] made use of the k-Gaifman theory in their proof of the $0-1$ law for $L^\omega_{\infty\omega}$. We make the following easy observation.

Proposition 11. *Let $A \models \Gamma_k$, and let B be any (finite or infinite) model. Then $B \preceq^k_{\infty\omega} A$. Equivalently, for all $\varphi \in L^k_{\infty\omega}(\exists)$, if φ is satisfiable, then $A \models \varphi$.*

Proof. The proof follows easily from Proposition 8 by considering the eternal $\exists^k$-game on B and A with the Duplicator playing on A. The k-Gaifman axioms essentially say that D can extend a partial isomorphism with domain of size $< k$ in every possible way. Therefore, she has a winning strategy for the game. ■

We observe that this result yields a compactness theorem over finite structures and a finitary analog of the Löwenheim-Skolem Theorem for $L^k_{\infty\omega}(\exists)$.

Corollary 12. *For every $k \in \omega$, there is an $n_k \in \omega$ such that for every set Φ of sentences of $L^k_{\infty\omega}(\exists)$, Φ is satisfiable, if and only if, every finite subset of Φ is satisfiable, if and only if, Φ is satisfied in a model of size n_k.*

The next proposition establishes that there are finite structures whose $L^k(\exists)$-theory is not finitely axiomatizable in $L^k(\exists)$.

Proposition 13. *For all $k \geq 2$, there is a model $A_k \in \mathcal{F}$ such that the $L^k(\exists)$-theory of A_k is not finitely axiomatizable in $L^k(\exists)$.*

Proof. Let A_k be any finite model of the k-Gaifman theory over the language of graphs. We show that for any $n \in \omega$, there is a B^n_k such that $A_k \preceq^{k,n} B^n_k$ and $A_k \not\preceq^{k,n+1} B^n_k$. This implies that the theory of A_k cannot be axiomatized by $L^k(\exists)$ sentences of quantifier rank $\leq n$ and, therefore, that it is not finitely axiomatizable in $L^k(\exists)$.

For the purpose of defining the models B^n_k, we require the following notion and notation. A basic k-type π satisfies the *distinctness condition* if for every

$l < k$, the formula $x_l \neq x_k \in \pi$. Let $\{\pi_1, \ldots, \pi_s\}$ be a set of basic $(k-1)$-types such that

1. every basic $(k-1)$-type is equivalent to some π_i and
2. if $i \neq j$, then π_i is not equivalent to π_j.

Similarly, for each $1 \leq i \leq s$, let $\{\pi_{i,1}, \ldots, \pi_{i,n(i)}\}$ be a set of basic k-types each of which extends π_i and satisfies the distinctness condition such that

1. every basic k-type which extends π_i and satisfies the distinctness condition is equivalent to some $\pi_{i,j}$ and
2. if $j \neq j'$, then $\pi_{i,j}$ is not equivalent to $\pi_{i,j'}$.

We proceed to define the models B^n_k. Let B^1_k be the graph on two vertices with exactly one loop and no other edges. Thus B^1_k realizes both basic 1-types. Given that B^n_k has been defined, we now define B^{n+1}_k as an extension of B^n_k. For each $(k-1)$-tuple $\overline{b}$ of elements of B^n_k, let $\tau(\overline{b})$ be the unique i such that $B^n_k \models \pi_i[\overline{b}]$, and let $X_{\overline{b}} = \{b^{n+1}_{\overline{b},j} \mid 1 \leq j \leq n(\tau(\overline{b}))\}$ be a set of distinct objects disjoint from B^n_k. We suppose that for any distinct pair of $(k-1)$-tuples $\overline{a}$ and $\overline{b}$ of elements of $B^n_k, X_{\overline{a}} \cap X_{\overline{b}} = \emptyset$. Let X be the union of all the sets $X_{\overline{b}}$. We let the universe of $B^{n+1}_k = B^n_k \cup X$. The edge relation of B^{n+1}_k is obtained from that of B^n_k by adding the minimal number of edges so that each k-tuple $\overline{b} * b^{n+1}_{\overline{b},j}$ satisfies $\pi_{\tau(\overline{b}),j}$. It is easy to see that each B^{n+1}_k is well-defined. We say that the *height* of an element b introduced in this construction is the least n such that $b \in B^n_k$.

We first show that $A_k \preceq^{k,n} B^n_k$. By Proposition 7, it suffices to describe a winning strategy for D in the n-round $\exists^k$-game with D playing on B^n_k and S playing on A_k. The strategy we describe for D will allow her to play her m^{th} move on some $b \in B^m_k$, for each $m \leq n$. In round 1, D answers the first move of S by playing her pebble on the appropriate element of $B^1_k \subseteq B^n_k$ to create a partial isomorphism. Suppose that D has played only onto elements of B^m_k through round m, where $m < n$. Let S choose pebble pair (α_l, β_l) to play in round $(m+1)$. We consider two cases. If S plays α_l on the same element as some $\alpha_{l'}$, for $l \neq l'$, then D must play β_l onto the element pebbled by $\beta_{l'}$. Doing so, she obviously maintains a partial isomorphism and succeeds in playing within B^{m+1}_k. On the other hand, suppose that S plays α_l on a distinct element such that the elements pebbled by $\overline{\alpha} * \alpha_l$ on A after the round satisfy $\pi_{i,j}$ (we may need to pad the tuple pebbled by $\overline{\alpha}$ to a tuple of length $(k-1)$ by repeating its last element, if all the pebbles are not in play at this round). Before D plays her $(m+1)^{st}$ move, the pebbles $\overline{\beta}$ are on a tuple $\overline{b}$ (similarly padded, if necessary) that satisfies π_i. She then plays β_l on the element $b^{m+1}_{\overline{b},j} \in B^{m+1}_k$, thereby maintaining a partial isomorphism. This strategy enables her to win the n-round game.

Next, we show that $A_k \not\preceq^{k,n+1} B^n_k$. By Proposition 7, it suffices to show that S can win the $(n+1)$-round game with D playing on B^n_k and S playing on A_k. We describe a strategy for play by S which forces D to pebble an element of height at least m by the end of round m to avoid losing at that round. It follows that S wins the $(n+1)$-round game since all elements of B_k^n have height $\leq n$. S plays as follows. He first places his k-pebbles on a set of k distinct elements which form a k-clique, that is, for every pair of distinct pebbled elements a and a', $A_k \models E(a, a')$. S may play in this way since $A_k \models \Gamma_k$. By our construction above, if $b, b' \in B_k^n$ are distinct elements of the same height, $B_k^n \not\models E(b, b')$. It follows immediately that any r-clique in B_k^n contains an element of height at least r. Therefore, if S has not won by round k, D has pebbled an element of height at least k by the end of that round. Note that in case $(n+1) \leq k$, we are done, since at round $(n+1)$, D will be unable to play onto an element of height at least $(n+1)$ to form an $(n+1)$-clique.

We proceed to describe the strategy for S's continuing play under the assumption that $k < (n+1)$. Suppose that through round $m, k \leq m < (n+1)$, D has played a pebble onto an element of height at least m, and that the k pebbles S has played lie on distinct elements of A_k which form a k-clique. We show how S can play to ensure that D must play onto an element of height at least $(m+1)$ at round $(m+1)$, if she is to prevent S from winning at this round, and leave the round with a k-clique pebbled. Suppose that β_i is pebbling an element b of height greater than the height of any other element pebbled in B_k^n at round m. By our hypothesis, the height of b is at least m. Pick $j \neq i$ (recall that $2 \leq k$) and let $a \in A_k$ be the element pebbled by α_j. S picks up α_j and places it on an $a' \in A_k$ such that

1. $A_k \models E(a, a) \leftrightarrow \neg E(a', a')$ and
2. for every $a'' \in A_k$ on which one of the remaining $(k-1)$ pebbles lies, $a' \neq a''$ and $A_k \models E(a', a'') \wedge E(a'', a')$.

The existence of such an a' follows from the fact that $A_k \models \Gamma_k$. We claim that to avoid losing at this round, D must play her pebble β_j onto an element b' of height greater than the height of b, and hence of height at least $(m+1)$. Let b'' be the element pebbled by β_j at round m. By our construction, each element of B_k^n is connected to at most $(k-1)$ elements of lesser height. Therefore, from the hypotheses that S had pebbled a k-clique at round m, and that b is an element of maximal height pebbled by D at that round, we may conclude that the only element of height $\leq$ the height of b adjacent to b onto which D could play β_j is b'' itself. But this play would fail to maintain a partial isomorphism with the elements S has now pebbled at round $(m+1)$ by the first condition we have imposed on the choice of a' above. Therefore, to avoid losing at round $(m+1)$, D must pebble an element of height at least $(m+1)$. ■

The next result follows immediately.

Corollary 14. *There are infinitely many formulas of $L^k(\exists)$ which are pairwise inequivalent over $\mathcal{F}$.*

We now consider $L^k_{\infty\omega}(\exists)$-theories and normal forms for $L^k_{\infty\omega}(\exists)$ sentences over $\mathcal{F}$. We let $\mathrm{Th}^k_{\exists}(A)$ denote the $L^k_{\infty\omega}(\exists)$-theory of A. Before proceeding, we define the following fragments of $L^k_{\infty\omega}(\exists)$.

1. Let $\bigwedge L^k(\exists) = \{\theta \mid \theta = \bigwedge \Phi, \text{ for some } \Phi \subseteq L^k(\exists)\}$.
2. Let $\bigvee L^k(\exists) = \{\theta \mid \theta = \bigvee \Phi, \text{ for some } \Phi \subseteq L^k(\exists)\}$.
3. Let $\bigwedge(\bigvee L^k(\exists)) = \{\theta \mid \theta = \bigwedge \Phi, \text{ for some countable } \Phi \subseteq \bigvee L^k(\exists)\}$.
4. Let $\bigvee(\bigwedge L^k(\exists)) = \{\theta \mid \theta = \bigvee \Phi, \text{ for some countable } \Phi \subseteq \bigwedge L^k(\exists)\}$.

Proposition 15. *For all finite structures A, there is a $\theta \in \bigwedge L^k(\exists)$ such that* $\mathrm{Mod}_f(\theta) = \mathrm{Mod}_f(\mathrm{Th}^k_\exists(A))$.

Proof. Observe that $\mathrm{Mod}_f(\mathrm{Th}^k_\exists(A)) = \{B \in \mathcal{F} \mid A \preceq^k_{\infty\omega} B\}$. Let $\mathcal{C}_A = \mathcal{F} - \mathrm{Mod}_f(\mathrm{Th}^k_\exists(A))$. By Proposition 9, for each $B \in \mathcal{C}_A$, there is a sentence $\varphi_B \in L^k(\exists)$ such that $A \models \varphi_B$ and $B \not\models \varphi_B$. Let $\theta = \bigwedge_{B \in \mathcal{C}_A} \varphi_B$. It is easy to verify that $\mathrm{Mod}_f(\theta) = \mathrm{Mod}_f(\mathrm{Th}^k_\exists(A))$. ■

Kolaitis and Vardi [12] obtained a normal form for the negation free fragment of $L^k_{\infty\omega}(\exists)$ over $\mathcal{F}$. It is easy to extend their result to $L^k_{\infty\omega}(\exists)$ and to provide a dual normal form as well. We codify these normal forms in the next proposition.

Proposition 16 (Kolaitis and Vardi [12]). *For each $\varphi \in L^k_{\infty\omega}(\exists)$, there is a $\theta \in \bigvee(\bigwedge L^k(\exists))$ and a $\zeta \in \bigwedge(\bigvee L^k(\exists))$ such that* $\mathrm{Mod}_f(\varphi) = \mathrm{Mod}_f(\theta) = \mathrm{Mod}_f(\zeta)$.

Proof. Let $\mathcal{C} = \mathrm{Mod}_f(\varphi)$. By Proposition 10, for each $A \in \mathcal{C}, B \in \mathcal{F} - \mathcal{C}$, there is a sentence $\theta_{A,B} \in L^k(\exists)$ such that $A \models \theta_{A,B}$ and $B \not\models \theta_{A,B}$. Let $\theta = \bigvee_{A \in \mathcal{C}}(\bigwedge_{B \notin \mathcal{C}} \theta_{A,B})$ and let $\zeta = \bigwedge_{B \notin \mathcal{C}}(\bigvee_{A \in \mathcal{C}} \theta_{A,B})$. It is easy to verify that the proposition holds for this choice of θ and ζ. ■

Next we show that the fragments $\bigwedge L^k(\exists)$ and $\bigvee L^k(\exists)$ are closed under finite conjunction, finite disjunction, and existential quantification over $\mathcal{F}$. This means that if an $L^k_{\infty\omega}(\exists)$-definable query cannot be expressed in either $\bigwedge L^k(\exists)$ or $\bigvee L^k(\exists)$, then it is only definable using both an infinitary conjunction and an infinitary disjunction.

Proposition 17. *The languages $\bigwedge L^k(\exists)$ and $\bigvee L^k(\exists)$ are both closed under finite conjunction, finite disjunction, and existential quantification over $\mathcal{F}$.*

Proof. Let $\Phi = \{\varphi_i(x, \overline{y}) \mid i \in \omega\}$ be a set of formulas of $L^k(\exists)$. We show that if $\theta(\overline{y}) = \exists x \bigwedge \Phi$, then $\theta(\overline{y})$ is equivalent over $\mathcal{F}$ to some formula $\theta'(\overline{y}) \in \bigwedge L^k(\exists)$. (The other closure conditions may be easily verified.) Let $\psi_m = \bigwedge_{0 \leq l \leq m} \varphi_l(x, \overline{y})$ and let $\theta'(\overline{y}) = \bigwedge_{m \in \omega} \exists x \psi_m$. We show θ' is equivalent to θ. It is obvious that θ implies θ'. Let $A \in \mathcal{F}$ and $\overline{a} \in A$ be such that $A \models \theta'[\overline{a}]$. Because A is finite, there is some $a' \in A$ such that for arbitrarily large m, $A \models \psi_m[a', \overline{a}]$. Therefore $A \models \bigwedge_{m \in \omega} \psi_m[a', \overline{a}]$, and θ' implies θ. ■

Below we show that the query classes $\bigwedge L^k(\exists)$ and $\bigvee L^k(\exists)$ are proper subsets of $\bigwedge(\bigvee L^k(\exists))$ and that neither of $\bigwedge L^k(\exists)$ and $\bigvee L^k(\exists)$ is a subset of the other. We first give necessary and sufficient conditions for classes to be definable in $\bigwedge L^k(\exists)$ and $\bigvee L^k(\exists)$.

Proposition 18. *1. A class $\mathcal{C}$ is definable in $\bigwedge L^k(\exists)$ iff for all $B \notin \mathcal{C}$, there is a $\varphi_B \in L^k(\exists)$ such that $B \not\models \varphi_B$ and for all $A \in \mathcal{C}, A \models \varphi_B$.*

2. A class $\mathcal{C}$ is definable in $\bigvee L^k(\exists)$ iff for all $A \in \mathcal{C}$, there is a $\varphi_A \in L^k(\exists)$ such that $A \models \varphi_A$ and for all $B \notin \mathcal{C}, B \not\models \varphi_A$.

Proof. To prove 1., suppose that $\mathcal{C}$ is defined by the sentence $\bigwedge_{n\in\omega} \psi_n$, and that $B \notin \mathcal{C}$. Then there is some ψ_m such that $B \not\models \psi_m$. Let φ_B be this ψ_m. In the other direction, observe that the sentence $\varphi = \bigwedge_{B\notin\mathcal{C}} \varphi_B$ defines $\mathcal{C}$. The proof of 2. is similar. ■

Proposition 19. *For each $k \geq 2$, there is a polynomial time computable boolean query $\mathcal{C} \in \bigwedge L^k(\exists) - \bigvee L^k(\exists)$.*

Proof. Let $k \geq 2$ be given and let the graph A_k be a model of the k-Gaifman theory. Let T be the $L^k(\exists)$-theory of A_k and let $\theta = \bigwedge T$. Clearly, $\theta \in \bigwedge L^k(\exists)$. Let $\mathcal{C} = \mathrm{Mod}_f(\theta)$. It is easy to see that $\mathcal{C} = \{B \in \mathcal{F} \mid A_k \preceq^k B\}$. It then follows immediately from the fact that the relation $\preceq^k$ is polynomial time computable (see Kolaitis and Vardi [12]), that $\mathcal{C}$ is polynomial time computable. In the proof of Proposition 13, we showed that for every satisfiable $\varphi \in L^k(\exists), \mathrm{Mod}_f(\varphi) \not\subseteq \mathcal{C}$. It follows immediately that $\mathcal{C} \neq \mathrm{Mod}_f(\psi)$ for every sentence $\psi \in \bigvee L^k(\exists)$. ■

Proposition 20. *There is a polynomial time computable boolean query $\mathcal{C} \in \bigvee L^2(\exists)$ such that for all $k \in \omega$, $\mathcal{C} \notin \bigwedge L^k(\exists)$. In consequence, for each $k \geq 2$, there is a class $\mathcal{C} \in \bigvee L^k(\exists) - \bigwedge L^k(\exists)$.*

Proof. Over the signature $\sigma = \{E, s, t\}$, let $\mathcal{C} = \{A \mid$ there is a path from s to $t\}$, the class of (s,t)-connected graphs. This class is clearly in $\bigvee L^2(\exists)$. As noted earlier, it is in Datalog, and, hence, polynomial time computable. From Proposition 18, to show that $\mathcal{C} \notin \bigwedge L^k(\exists)$, it suffices to show that there is a $B \notin \mathcal{C}$ such that for all $n \in \omega$, there is an $A_n \in \mathcal{C}$ such that $A_n \preceq^{k,n} B$. This latter condition is equivalent to D's possessing a winning strategy for the n-round $\exists^k$-game on A_n and B. We construct B to give her the greatest possible freedom in choosing her moves. Let M be any graph such that $M \models \Gamma_{k+1}$, and let M_s (resp. M_t) be obtained from M by requiring that s (resp. t) denote a loop-free element. We define B to be the disjoint union of M_s and M_t, thus insuring that $B \notin \mathcal{C}$.

For each n, let A_n be the simple chain from s to t of length 2^{n+2}. The basic idea is that by choosing the chain to be long enough, S will not be able to witness the existence of a path from s to t in only n moves. Let $d(x,y)$ be the natural distance function on A_n.

We now describe D's strategy. In each round m, D chooses to play on an element of M_s iff S just played a pebble on $a \in A_n$ such that either (i) $d(s,a) \leq 2^{(n+2)-m}$; or (ii) there is a j such that β_j is on an element of M_s and $d(\alpha_j, a) \leq 2^{(n+2)-m}$. She then plays her pebble on an element of the appropriate component of B so that she maintains a partial isomorphism among the pebbles on that component. It is easy to see that this is possible because M_s and M_t are models of Γ_{k+1}.

In order to establish that this is a winning strategy, it suffices to verify the following two claims.

1. In each round $l \leq n$, if D plays a pebble β_i on M_s, then α_i is not adjacent to t on A_n. Similarly for M_t and s.
2. After each round l, for all pairs of pebbles $\{\alpha_i, \alpha_j\}$, if $A_n \models E(\alpha_i, \alpha_j)$, then β_i and β_j are on the same component of B.

We argue, by induction, that if D plays β_i on M_s in round m, then $d(s, \alpha_i) \leq (2^{(n+2)-1} + 2^{(n+2)-2} + \ldots + 2^{(n+2)-m}) < 2^{n+2} - 1$. Since $d(s,t) = 2^{n+2}$, this establishes that $A_n \not\models E(\alpha_i, t)$. In round 1, D plays β_i on M_s iff $d(s, \alpha_i) \leq 2^{(n+2)-1}$. Suppose that in round $m+1$ D plays β_i on M_s. Then either $d(s, \alpha_i) \leq 2^{(n+2)-m}$ or there is an α_j such that β_j is on M_s, $d(\alpha_i, \alpha_j) \leq 2^{(n+2)-(m+1)}$, and, by induction hypothesis, $d(s, \alpha_j) \leq (2^{(n+2)-1} + 2^{(n+2)-2} + \ldots + 2^{(n+2)-m})$. In both cases, the induction condition is maintained. The second part of Claim 1 follows from the fact that in round m, if D plays β_i on M_t, then S must have played α_i such that $d(s, \alpha_i) > 2^{(n+2)-m} > 1$. To prove Claim 2, observe that at each round m, if $\beta_i \in M_s$, and $\beta_j \in M_t$, then $d(\alpha_i, \alpha_j) \geq 2^{(n+2)-m} > 1$. The details are similar to the previous argument. ■

The next result shows that the normal form for $L^k_{\infty\omega}(\exists)$ over $\mathcal{F}$ given in Proposition 16 is optimal.

Proposition 21. *For all $k \geq 2$, there is a class $\mathcal{C} \subseteq \mathcal{F}$ such that $\mathcal{C} \in \bigvee(\bigwedge L^k(\exists)) - (\bigwedge L^k(\exists) \cup \bigvee L^k(\exists))$.*

Proof. The proof of this proposition is a synthesis of the proofs of the preceding two results. We define a set of models $\{A_1, A_2, \ldots\}$ which are pairwise $L^k(\exists)$-incompatible such that for each i, the $L^k(\exists)$-theory of A_i is not finitely axiomatizable in $L^k(\exists)$. We then let $\mathcal{C} = \{B \mid \exists i(A_i \preceq^k B)\}$. The arguments to show that this class is neither in $\bigvee L^k(\exists)$ nor in $\bigwedge L^k(\exists)$ are minor variants of the proofs of Propositions 19 and 20.

We define each model A_i as an expansion of a homeomorphic image of a graph which is a model of the $(k+1)$-Gaifman theory. Let R be a finite graph that satisfies Γ_{k+1}; observe that R also verifies Γ_k. Each A_i is obtained from R by replacing all edges which are not loops by pairwise disjoint paths of length i. Where there is a two-way, undirected edge, a single undirected path is inserted, rather than two directed paths. To clarify the exposition, we also add a unary predicate V to the signature to label the original 'vertices' of R.

To verify that $\mathcal{C}$ is not in $\bigvee L^k(\exists)$, it suffices to show that there is a model $A \in \mathcal{C}$ and a sequence $B^1, B^2, \ldots$, disjoint from $\mathcal{C}$, such that for each n, $A \preceq^{k,n} B^n$. Let A be A_1, and let each B^n be obtained from the model B^n_k from the proof of Proposition 13 by putting every element into the extension of the predicate V. From that proof it is immediate that, for all n, $A_1 \preceq^{k,n} B^n$ but $A_1 \not\preceq^k B^n$. For all $2 \leq i$, $A_i \models \exists x \neg Vx$ and, consequently, $A_i \not\preceq^k B^n$. This establishes that each B^n is not in $\mathcal{C}$.

In order to show that $\mathcal{C} \notin \bigwedge L^k(\exists)$, we now define a single $B' \notin \mathcal{C}$ such that for all n, there is an $A_{f(n)}$ such that $A_{f(n)} \preceq^{k,n} B'$. By Proposition 18, this will establish that $\mathcal{C} \notin \bigwedge L^k(\exists)$. Let R^+ be an expansion of R obtained by labeling exactly one looped element with the predicate V; and let R^- be obtained similarly by labeling a loop-free element. Here the predicate V plays the same role as the constants s and t in the proof of Proposition 20. We define B' to be the disjoint union of k copies of both R^+ and R^-, and let $f(x) = 2^{x+2}$. It is easy to see that $B' \notin \mathcal{C}$. As in the proof of Proposition 20, the Duplicator wins the n-move $\exists^k$-game on A_{2^n+2} and B' because the labeled vertices of A_{2^n+2} are too far apart for S to distinguish the models by witnessing that they are actually connected. ■

Finally, we prove the following separation.

Proposition 22. *Over $\mathcal{F}$, for $k \geq 3, L^k(\exists) \subset (\bigwedge L^k(\exists)) \cap (\bigvee L^k(\exists))$.*

Proof. Let Path(x, y) express the binary query 'there is an E-path from x to y.' For signature $\sigma = \{E, s\}$, we define $\mathcal{C} = \{A \mid \exists x($ Path(s, x) and Path$(x, x))\}$. Let $\theta_n(x, y)$ be an $L^3(\exists)$ formula that defines the binary query 'there is a path of length n from x to y.' It is easy to see that $\mathcal{C}$ is in $\bigvee L^k(\exists)$. Also observe that $\varphi = \bigwedge_{n \in \omega} \exists x \exists y(s = x \wedge \theta_n(x, y))$ defines $\mathcal{C}$. Finally, there are arbitrarily large minimal models in $\mathcal{C}$, that is, models $A \in \mathcal{C}$ such that for all proper submodels $B \subset A, B \notin \mathcal{C}$. This immediately implies that $\mathcal{C} \notin$ FO$(\exists)$ and, *a fortiori*, not in $L^k(\exists)$. ■

4 The Failure of Existential Preservation for $L^\omega_{\infty\omega}$

In this section we prove that $L^\omega_{\infty\omega} \cap \text{EXT} \not\subseteq L^\omega_{\infty\omega}(\exists)$. Indeed, we establish that there is a sentence $\theta \in L^\omega_{\infty\omega}$ such that Mod(θ) is closed under extensions, but there is no $\psi \in L^\omega_{\infty\omega}(\exists)$ such that Mod$_f(\theta) =$ Mod$_f(\psi)$. Thus, θ witnesses the failure of existential preservation for $L^\omega_{\infty\omega}$ simultaneously over the class of finite structures and over the class of all structures. The central lemma on which this result relies is of interest in itself. It says that for all $k \geq 3$, the finitary language L^k fails in a strong way to satisfy an existential preservation property. Andreka, van Benthem, and Nemeti [3] showed that for every $k \geq 3$, there is a sentence $\varphi_k \in L^k$ which is preserved under extensions, but which is not equivalent to any sentence of $L^k(\exists)$. For $k \geq 3$, the sentence φ_k they construct uses a relation symbol of arity $k - 1$ and has the property that it is equivalent to a sentence of $L^{k+1}(\exists)$. They state the following open problems.

- For any $k \geq 3$ and $n \in \omega$, find sentences $\varphi_n \in L^k$ which are preserved under extensions, but which are not equivalent to any sentence of $L^{k+n}(\exists)$.
- For $k > 3$, is there a formula of L^k containing only (one) binary relation symbols which is preserved under extensions, but is not equivalent to any sentence of $L^k(\exists)$?

The next proposition settles both these open problems. The main result of the section follows easily from the proof of this proposition.

Proposition 23. *For each $k < \omega$, there is a sentence $\theta_k \in L^3$, containing a single binary relation, such that*

1. $\text{Mod}(\theta_k)$ *is closed under extensions, but*
2. $\text{Mod}_f(\theta_k) \neq \text{Mod}_f(\varphi)$ *for all* $\varphi \in L^k(\exists)$.

Proof. Before presenting the full proof, we sketch the basic outline. Let the *k-pyramid* of B, $\mathcal{P}^k(B)$, be the smallest class of (finite and infinite) models containing B that is closed under substructures and L^k-equivalence. For each $k \geq 3$, we define finite structures A_k and B_k with the following properties:

1. $A_k \preceq^k_{\infty\omega} B_k$;
2. $\mathcal{P}^3(B_k)$ is L^3-definable;
3. $A_k \notin \mathcal{P}^3(B_k)$.

Let $\varphi_k \in L^3$ be such that $\text{Mod}(\varphi_k) = \mathcal{P}^3(B_k)$, and let $\theta_k = \neg\varphi_k$. It is obvious that $\text{Mod}(\theta_k)$ is closed under extensions, that $A_k \models \theta_k$, and that $B_k \not\models \theta_k$. Suppose $\varphi \in L^k(\exists)$ is such that $A_k \models \varphi$. Since $A_k \preceq^k_{\infty\omega} B_k$, this implies that $B_k \models \varphi$, and therefore that φ is not equivalent to θ_k.

We define structures A_k and B_k in terms of simpler submodels. For $f \leq t$, let the $[t, f]$-*flag*, $F[t, f]$, be the directed chain of length t with one additional vertex attached to the f^{th} link. That is, the vertex set of $F[t, f]$ is $\{0, 1, \ldots, t, t+1\}$, and the edge relation is $\{(i, i+1) \mid i < t\} \cup \{(f, t+1)\}$. A_k is the disjoint union of the $k+1$ flags—$F[2k+2, k+1], F[2k+2, k+2], \ldots, F[2k+2, 2k+1]$. Let the $[k, j]$-*tree*, $T[k, j]$, be the tree obtained from A_k by fusing the i^{th} nodes of each flag, for all $i \leq j$. This tree has height $2k+2$ and the node at height j has outdegree $k+1$. Then B_k is the disjoint union of the k trees— $T[k, 0], T[k, 1], \ldots, T[k, k-1]$.

First we show that $A_k \preceq^k_{\infty\omega} B_k$ by describing a winning strategy for D in the eternal $\exists^k$-game on A_k and B_k. A *component* of a model is a maximal connected submodel. Observe that every component of A_k is embeddable in every component of B_k. Call a component of either A_k or B_k *vacant* at round n if there is no pebble located on any element of that component before the players make their n^{th} moves. We consider two cases of moves for S. First, suppose that in some round n, S plays pebble α_i on a vacant component A^n of A_k. Since there are only k pairs of pebbles, and since pebble β_i is not on the board, there is a vacant component B^n of B_k, and an isomorphic injection $h_n : A^n \mapsto B^n$. D will play pebble β_i on $h_n(\alpha_i)$. In the other case, S plays on a non-vacant component A^n. There is some $m < n$ such that A^n has been occupied continuously since round m and either $m = 1$ or A^n was vacant at round $m - 1$. Thus $A^n = A^m$, and there are previously defined B^m and h_m. D now plays β_i on $h_m(\alpha_i)$. By this condition, every pair of pebbles (α_l, β_l) on components A^m and B^m satisfies the condition that $h_m(\alpha_l) = \beta_l$. In both cases, it is clear that D has maintained a partial isomorphism. By Proposition 8, it now follows immediately that $A_k \preceq^k_{\infty\omega} B_k$.

Next, we show that $\mathcal{P}^3(B_k)$ is definable in L^3. Consider the following properties:

1. A contains no chains of length $\geq 2k+2$.
2. A contains no cycles of length $\leq 2k+2$.

3. No element $a \in A$ has indegree ≥ 2, that is, $A \models \neg\exists x \exists y \exists z(x \neq y \wedge Exz \wedge Eyz)$.

It is easy to show that each property is expressible in L^3, is closed under substructures, and holds of B_k. From this it follows immediately that each $B' \in \mathcal{P}^3(B_k)$ possesses all three properties. Consequently, every member of $\mathcal{P}^3(B_k)$ is a forest consisting of directed trees of height $\leq 2k+2$.

Next we note the following facts:

Lemma 24. *Let A and B be the disjoint unions of components $\{A_1, \ldots, A_m\}$ and $\{B_1, \ldots, B_n\}$, respectively. For $k \geq 3$, $A \equiv^k_{\infty\omega} B$ if and only if for each component A_i $[B_i]$, either the number of components of A that are L^k-equivalent to it is equal to the number of components of B that are L^k-equivalent to it or both numbers are $\geq k$.*

This result can be proved by a simple pebble game argument.

Lemma 25. *For each h, and each $k \geq 3$, up to equivalence in L^k there are only finitely many trees of height $\leq h$.*

The proof proceeds by induction on h. The case where $h = 1$ is obvious. Given a tree T, call a proper subtree that contains a node t of height 1 and all of its descendents a 1-*tree* of T. For $h > 1$, we claim that two trees T_1 and T_2 of height at most h are L^k-equivalent if and only if for each 1-tree $T' \subset T_i$, the number of 1-trees of T_1 that are L^k-equivalent to T' equals the number of 1-trees of T_2 that are L^k-equivalent to T', or both numbers are $\geq k$. The argument is just like the proof of the preceding lemma. From the claim, the lemma follows immediately.

Corollary 26. *For each h, and each $k \geq 3$, up to equivalence in L^k there are only finitely many forests of height $\leq h$.*

This is an immediate consequence of the preceding lemmas.

These observations establish that there are only finitely many complete L^k-theories that are satisfiable in $\mathcal{P}^3(B_k)$. Moreover, each such theory has a finite model. By [5], every such theory is axiomatized by a single L^k sentence. Hence, if we let φ_k be the disjunction of these sentences, we have $\mathrm{Mod}(\varphi_k) = \mathcal{P}^3(B_k)$ as desired.

Finally, we argue that $A_k \notin \mathcal{P}^3(B_k)$. By the definition of $\mathcal{P}^3(B_k)$, for every $B' \in \mathcal{P}^3(B_k)$, there is an $m \in \omega$ and a sequence $(E_0, D_1, E_1, \ldots, D_m, E_m)$ of structures, with $B_k = E_0$ and $B' = E_m$, such that:

1. For all $1 \leq i \leq m, D_i \subseteq E_{i-1}$.
2. For all $1 \leq i \leq m$, $D_i \equiv^3 E_i$.

It suffices to show that for any such sequence, A_k cannot be embedded in any E_i. Let $g : \mathcal{P}^3(B_k) \mapsto \{0, 1, \ldots, k+1\}$ be the function such that $g(D)$ is the maximum number of components of A_k that can be embedded in D pairwise disjointly. We show that for each $i \leq m, g(E_i) < k+1$. In fact, we show that g is monotonically decreasing on the aforementioned sequence. Because each D_i is a submodel of E_{i-1}, it is clear that $g(D_i) \leq g(E_{i-1})$. It remains to establish that $g(B_k) < k+1$ and that $g(E_i) \leq g(D_i)$.

Observe that any embedding of a flag $F[2k+2, f]$ into a component C of any $B' \in \mathcal{P}^3(B_k)$ must map the root of the flag to the root of C. This implies that no two flags of A_k can be disjointly embedded into any such component and, since B_k has only k components, that $g(B_k) < k+1$.

From Lemma 24, it follows that every E_i can be obtained from D_i by repeated application of the following three operations. First, replace some component with a component that is L^3-equivalent to it. Second, add a disjoint copy of a tree that is L^3-equivalent to at least 3 components. Third, remove a component that is L^3-equivalent to at least 3 other components. Thus, it suffices to argue that no such operation performed on some $B' \in \mathcal{P}^3(B_k)$ can yield a B'' such that $g(B'') > g(B')$. It is obvious that removing a component cannot increase the value of g.

We claim that it suffices to consider the effect of the other two operations on components of height $= 2k+2$. If trees T and T' are L^3-equivalent, then they have the same height. Also, no component $F[2k+2, f]$ of A_k can be embedded in any tree of height $< 2k+2$. This establishes that the presence of shorter components in a model B does not affect the value of $g(B)$.

Observe that for all trees T and T' such that $T \equiv^3 T'$, $F[t, f]$ can be embedded in T iff it can be embedded in T'. This is because the following property can be expressed in L^3: there is an element x such that (i) there is a y such that there is a path of length f from y to x; (ii) x has outdegree 2; (iii) there is a y such that there is a path of length $t-f$ from x to y. Over trees, this property says that the model embeds $F[t, f]$. Consequently the operation of replacement cannot increase the value of g.

It remains to establish that adding an additional component to a model $B' \in \mathcal{P}^3(B_k)$ does not change the value of g. We observe that B_k has the following properties:

1. For each $(2k+2)$-chain contained in B_k there is at most one $j, 0 \leq j \leq k-1$, such that the j^{th} link of the chain has outdegree > 1.
2. For each $(2k+2)$-chain contained in B_k there is at most one $j, k+1 \leq j \leq 2k+1$, such that the j^{th} link of the chain has outdegree > 1.

These properties are closed under substructures and L^3-equivalence; consequently, they hold of every model $B' \in \mathcal{P}^3(B_k)$. Let C_1, C_2, and C_3 be L^3-equivalent components of B' of height $2k+2$. The above argument establishes that each C_i is either some $F[2k+2, f]$, or the simple $(2k+2)$-chain. Let B'' be the extension of B' obtained by adding a component C_4. Observe that, in fact, all four components must be isomorphic, and embed at most one isomorphism type of flag. Therefore, the image of any embedding $h : A_k \mapsto B''$ can contain vertices from at most one of these four components. This demonstrates that $g(B') = g(B'')$, and completes the proof. ■

The following result establishes the failure of existential preservation for $L^{\omega}_{\infty\omega}$.

Theorem 27. *There is a sentence $\theta \in L^{\omega}_{\infty\omega}$ such that both*

1. $\mathrm{Mod}(\theta)$ *is closed under extensions.*
2. *For all* $\varphi \in L^{\omega}_{\infty\omega}(\exists), \mathrm{Mod}_f(\theta) \neq \mathrm{Mod}_f(\varphi)$.

Proof. We claim that it suffices to show that for each $k \in \omega$ there is a sentence $\theta_k \in L^3$ and a pair of finite models A_k and B_k such that

1. $\mathrm{Mod}(\theta_k)$ is closed under extensions.
2. $A_k \models \theta_k$ and $B_k \not\models \theta_k$.
3. $A_k \preceq^k_{\infty\omega} B_k$.
4. For all $j, A_j \models \theta_k$.

Let $\theta = \bigwedge_k \theta_k$. It is clear that θ is closed under extensions and that it has finite models, since it is true in each A_k. Suppose that φ is a sentence in $L^k_{\infty\omega}(\exists)$ such that θ implies φ. Then $A_k \models \varphi$, and therefore $B_k \models \varphi$. But for all l, $B_l \not\models \theta$. Therefore, $\mathrm{Mod}_f(\theta) \neq \mathrm{Mod}_f(\varphi)$.

The sentences θ_k and the models A_k and B_k from the proof of Proposition 23 fail to meet condition 4 because for $j < k$, $A_j \not\models \theta_k$. To see this, observe that A_j will always be a submodel of B_k. To fix this defect, it suffices to construct A'_k, B'_k, and θ'_k as in the proof of Proposition 23 that also satisfy the additional condition that, for all j and k, $A'_j \notin \mathcal{P}^3(B'_k)$. In order to accomplish this, we add simple 'gadgets' to the models. Let the *k-cycle*, C_k, be the graph on k vertices whose edge relation forms a simple, directed cycle of length k. Then let A'_k and B'_k be obtained from A_k and B_k, respectively, by adding a disjoint copy of C_k. By slightly modifying the proof of Proposition 23, we can show that $A'_k \preceq^k_{\infty\omega} B'_k$, and that there is a $\theta'_k \in L^3$ satisfied by exactly the models in the complement of $\mathcal{P}^3(B'_k)$ such that $A'_k \models \theta'_k$. Finally, it is easy to verify that for $j \neq k$, the j-cycle cannot be embedded in any $B \in \mathcal{P}^3(B'_k)$ and, therefore, $A'_j \models \theta'_k$. ■

5 Generalized Preservation Theorems in the Finite Case

In this section, we prove some generalized preservation theorems for fragments of FO. Our results are of the form

$$L \cap \mathrm{EXT} \subseteq L'$$

for certain quantifier prefix classes $L \subset \mathrm{FO}$ and $L' = L^{\omega}_{\infty\omega}(\exists)$ or Datalog$(\neq, \neg)$. Recall that Tait [18] showed

$$\mathrm{FO} \cap \mathrm{EXT} \not\subseteq \mathrm{FO}(\exists),$$

and that Gurevich and Shelah [9, 10] gave examples showing that

$$\mathrm{FO}[\forall^*\exists^*] \cap \mathrm{EXT} \not\subseteq \mathrm{FO}(\exists).$$

Compton observed that

$$\mathrm{FO}[\exists^*\forall^*] \cap \mathrm{EXT} \subseteq \mathrm{FO}(\exists),$$

which shows that these examples are best possible in terms of quantifier alternation prefix (see [9]). Kolaitis and Vardi (see [2]) observed that the example of Gurevich and Shelah [9] can be defined in Datalog($\neq, \neg$). Theorem 29 below establishes that

$$\mathrm{FO}[\exists^*\forall\exists] \cap \mathrm{EXT} \subseteq \mathrm{Datalog}(\neq, \neg).$$

It follows that all the examples in the literature witnessing the failure of the Los-Tarski Theorem in the finite case are definable in Datalog($\neq, \neg$), since all these examples are in the prefix class FO$[\exists^*\forall\exists]$ (a sequence of existential quantifiers followed by one universal quantifier followed by one existential qunatifier). The next theorem establishes a slightly more general result with $L^{\omega}_{\infty\omega}(\exists)$ in place of Datalog($\neq, \neg$).

Theorem 28. $\mathrm{FO}[\exists^*\forall\exists^*] \cap \mathrm{EXT} \subseteq L^{\omega}_{\infty\omega}(\exists)$.

Proof. Let $\varphi \in \mathrm{FO}[\exists^*\forall\exists^*] \cap \mathrm{EXT}$. That is, $\varphi \in \mathrm{FO}[\exists^*\forall\exists^*]$ and $\mathrm{Mod}_f(\varphi) \in \mathrm{EXT}$. Let $\mathcal{C} = \mathrm{Mod}_f(\varphi)$. We proceed to show that $\mathcal{C} \in L^{\omega}_{\infty\omega}(\exists)$. By Proposition 10, it suffices to show that there is a k such that, for each $A \in \mathcal{C}$ and $B \notin \mathcal{C}$, there is a $\theta_{A,B} \in L^{k}_{\infty\omega}(\exists)$ such that $A \models \theta_{A,B}$ and $B \not\models \theta_{A,B}$.

Let $\varphi = \exists x_1 \ldots x_i \forall y \exists z_1 \ldots z_j \psi(\overline{x}, y, \overline{z})$, where ψ is quantifier free, and let $k = i+j+1$ (we suppose, without loss of generality, that $i > 0$). We now describe a winning strategy for S in the eternal $\exists^k$-game on A and B, for $A \in \mathcal{C}$ and $B \notin \mathcal{C}$, which establishes, by Proposition 8, the existence of $\theta_{A,B} \in L^{k}_{\infty\omega}(\exists)$ with the desired properties. There are two stages. Let $\overline{a} = (a_1, \ldots, a_i)$ be a sequence of elements of A such that $A \models \forall y \exists \overline{z} \psi(\overline{a}, y, \overline{z})$. If D has not lost after h rounds, for $h < i$, S plays pebble α_{h+1} on element a_{h+1}. If S has not won after i moves, and D has played her pebbles on $\overline{b} = (b_1, \ldots, b_i)$, then $B \models \exists y \forall \overline{z} \neg \psi(\overline{b}, y, \overline{z})$ (since $B \not\models \varphi$).

The goal of the second part of S's strategy is to force D to play a pebble on some element b' such that $B \models \forall \overline{z} \neg \psi(\overline{b}, b', \overline{z})$, without removing any of the pebbles $\alpha_1, \ldots, \alpha_i$ which 'fix the interpretation' of the variables $x_1, \ldots, x_i$ on both A and B. Regardless of the element a' on which S will have played his corresponding pebble, $A \models \exists \overline{z} \psi(\overline{a}, a', \overline{z})$, so that he can then win easily. In order to describe S's strategy, we first define a sequence of subsets of the universe of B. Let $\Gamma_0 = \{b' \mid b' \in B \text{ and } B \models \forall \overline{z} \neg \psi(\overline{b}, b', \overline{z})\}$. Observe that $B \models \exists y \forall \overline{z} \neg \psi(\overline{b}, y, z)$, and therefore Γ_0 is non-empty. Given $\Gamma_0, \ldots, \Gamma_m$, if $(\bigcup_{l \leq m} \Gamma_l) \cap \overline{b} = \emptyset$, then let B_{m+1} be the submodel of B whose universe is $(B - \bigcup_{l \leq m} \Gamma_l)$. Let $\Gamma_{m+1} = \{b' \mid b' \in B_{m+1} \text{ and } B_{m+1} \models \forall \overline{y} \neg \psi(\overline{b}, b', \overline{y})\}$. For each B_m, since $B_m \subseteq B$, we have that $B_m \models \forall \overline{x} \exists y \forall \overline{z} \neg \psi(\overline{x}, y, \overline{z})$. In particular, $B_m \models \exists y \forall \overline{z} \neg \psi(\overline{b}, y, \overline{z})$ and thus, as above, Γ_{m+1} is non-empty. Since B is finite, there is some n such that $\Gamma_n \cap \overline{b} \neq \emptyset$, and some element $b_f \in \Gamma_n \cap \overline{b}$ pebbled by β_f. Then B is partitioned into the sets $\Gamma_0, \ldots, \Gamma_{n-1}, B_n$. We also have that $A \models \exists \overline{z} \psi(\overline{a}, a_f, \overline{z})$, and $B_n \models \forall \overline{z} \neg \psi(\overline{b}, b_f, \overline{z})$.

The Spoiler can win by executing a substrategy that compels D to play in sets Γ_m of successively smaller index. Let $\overline{c}$ be a sequence of elements of length j such that, $A \models \psi(\overline{a}, a_f, \overline{c})$. S plays his next j moves on this sequence, until D makes a losing move or plays a pebble β_g onto an element in Γ_m, for $m \leq n-1$.

We claim that one of these two possibilities must occur. For suppose that D plays on a sequence $\overline{d} \subseteq B_n$. Then $B_n \models \neg\psi(\overline{b}, b_j, \overline{d})$, and $\psi(\overline{x}, y, \overline{z})$ witnesses that the function that takes $\overline{a} * a_j * \overline{c}$ to $\overline{b} * b_j * \overline{d}$ and preserves the denotations of constants is not a partial isomorphism.

Suppose that D has played some pebble β_g into some set Γ_m. By the same argument as above, reusing pebbles $\{\alpha_{i+1}, \ldots, \alpha_k\} - \{\alpha_g\}$, S can either win or force D to play into some $\Gamma_{m'}$, for some $m' < m$. Iterating this procedure, S can force D to play into Γ_0, and then win by using the same procedure one more time. ■

We remark the following two refinements of the foregoing theorem.

1. For each $B \notin \mathcal{C}$, there is a number m_B such that for all $A \in \mathcal{C}$, S wins the m_B-round $\exists^k$-game on A and B. (Here, m_B is determined by the maximum number of sets Γ that get defined on B, for any choice of D's first i moves.) It follows easily from Proposition 7 that this condition is equivalent to there being a $\theta_B \in L^k(\exists)$, with quantifier rank $\leq m_B$, such that for all $A \in \mathcal{C}$, $A \models \theta_B$, and $B \not\models \theta_B$. Then $\theta' = \bigwedge_{B \notin \mathcal{C}} \theta_B$ is equivalent to φ and is a single infinite conjunction of $L^k(\exists)$ sentences. We know by Proposition 20 that not all sentences of $L^k_{\infty\omega}(\exists)$ can be expressed in this form. Indeed, it follows from Theorem 29 below that if $\varphi \in \mathrm{FO}[\exists^*\forall\exists] \cap \mathrm{EXT}$, then φ is equivalent to a formula in $\bigwedge L^k(\exists) \cap \bigvee L^k(\exists)$ for some k.
2. Suppose that φ is an L^k sentence with quantifier type $\forall\exists^*$ (this notion of quantifier type may be defined straightforwardly, and is distinct from the notion of prefix class). In this case, we can show, by a modification of the proof of Theorem 28, that φ is equivalent to an $L^k_{\infty\omega}(\exists)$ sentence. This contrasts with Proposition 23 above which established that for all k, there is a sentence $\varphi_k \in L^3$ such that $\mathrm{Mod}_f(\varphi_k) \in \mathrm{EXT}$, but φ_k is not equivalent over $\mathcal{F}$ to any sentence in $L^k_{\infty\omega}(\exists)$.

Theorem 29. $\mathrm{FO}[\exists^*\forall\exists] \cap \mathrm{EXT} \subseteq \mathrm{Datalog}(\neq, \neg)$.

Proof. Let $\varphi = \exists x_1 \ldots x_j \forall y \exists z \beta(\overline{x}, y, z)$, with $\beta(\overline{x}, y, z)$ quantifier free. Let $\overline{c} = (c_1, \ldots, c_p)$ be the sequence of constants in the signature of φ and let $\mathcal{C} = \mathrm{Mod}_f(\varphi)$. For $a \in A$, we say that a *closes* with parameters $\overline{a}$ iff there is a sequence $a_0 (= a), a_1, \ldots, a_n$ such that for all $l < n$, $A \models \beta(\overline{a}, a_l, a_{l+1})$ and there is an $m \leq n$ such that $A \models \beta(\overline{a}, a_n, a_m)$. Note that this is equivalent to there being an a' such that there is a $\beta(\overline{a}, y, z)$-path from a to a', and a $\beta(\overline{a}, y, z)$-cycle including a'.

We claim that $A \models \varphi$ iff there is a j-tuple $\overline{a}$ such that every element of $\overline{a} \cup \overline{c}$ closes with parameters $\overline{a}$. Suppose that A does not satisfy these conditions. We prove that $A \models \forall\overline{x}\exists y \forall z \neg\beta(\overline{x}, y, z))$ where the latter sentence is equivalent to $\neg\varphi$. Let $\overline{a} \subseteq A$ be a sequence of length j. By hypothesis, there is an $a' \in \overline{a} \cup \overline{c}$ such that a' does not close with parameters $\overline{a}$. Since A is finite, this implies that there is an $m \geq 0$ and a sequence $a' = a'_0, \ldots, a'_m$ such that for all $l < m$, $A \models \beta(\overline{a}, a'_l, a'_{l+1})$ and $A \models \forall z \neg\beta(\overline{a}, a'_m, z)$, as desired.

In the other direction, let $\overline{a}$ be such that every member of $\overline{a} \cup \overline{c}$ closes with parameters $\overline{a}$. Let $\overline{s}_h = \langle a_{h0}(= a_h), \ldots, a_{hm_h} \rangle$ and $\overline{t}_h = \langle e_{h0}(= c_h), \ldots, e_{hn_h} \rangle$ be sequences witnessing that each element of $\overline{a} \cup \overline{c}$ closes with parameters $\overline{a}$. Let B be the submodel of A with universe $\bigcup_i \overline{s}_i \cup \bigcup_j \overline{t}_j$. Then it is easy to verify that $B \models \varphi$ and, since $\mathrm{Mod}_f(\varphi) \in$ EXT, it follows that $A \models \varphi$.

The following program, with $\overline{x} = (x_1, \ldots, x_j)$, computes φ:

$$\begin{aligned}
&P(\overline{x}, y, z) \longleftarrow \beta(\overline{x}, y, z)\\
&P(\overline{x}, y, z) \longleftarrow P(\overline{x}, y, w), P(\overline{x}, w, z)\\
&Q \longleftarrow P(\overline{x}, x_1, y_1), P(\overline{x}, y_1, y_1), \ldots, P(\overline{x}, x_j, y_j), P(\overline{x}, y_j, y_j),\\
&\qquad\quad P(\overline{x}, c_1, w_1), P(\overline{x}, w_1, w_1), \ldots, P(\overline{x}, c_p, w_p), P(\overline{x}, w_p, w_p)
\end{aligned}$$

This can be easily converted into a Datalog($\neq, \neg$) program. Let $\beta(\overline{x}, y, z) = \bigvee_i \delta_i$, where each δ_i is a conjunction of literals. Replace the clause $P(\overline{x}, y, z) \longleftarrow \beta(\overline{x}, y, z)$ with the clauses $P(\overline{x}, y, z) \longleftarrow \delta_i$, for all i. ■

6 Conclusion

In this section we discuss some open problems that are naturally suggested by our investigations and we present some further results bearing on the problem of preservation under homomorphisms in the finite case.

6.1 Open Problems

The first and most obvious question is the extent to which our results can be generalized from fragments of FO to the entire language. In this connection, we restate two of the problems mentioned earlier which remain open in light of our study.

Problem 1. Is FO $\cap$ EXT $\subseteq$ Datalog($\neq, \neg$)?

Problem 2. Is FO $\cap$ EXT $\subseteq L^{\omega}_{\infty\omega}(\exists)$?

Obviously, a positive answer to the first of these questions implies a positive answer to the second. Should the answer to these questions be negative, it would be of interest to characterize the classes FO $\cap$ Datalog($\neq, \neg$) and FO $\cap\, L^{\omega}_{\infty\omega}(\exists)$ in some informative way. An example of a characterization of this kind is the following theorem of Ajtai and Gurevich [2]. FO$^+(\exists)$ denotes the positive existential fragment of FO.

Proposition 30 (Ajtai and Gurevich [2]). FO $\cap$ Datalog $=$ FO$^+(\exists)$.

As remarked above, the Gurevich-Shelah counterexample to the Los-Tarski Theorem in the finite case witnesses that FO $\cap$ Datalog($\neq, \neg$) $\neq$ FO($\exists$). Might FO $\cap$ Datalog($\neq, \neg$) be contained in some level of the first-order quantifier alternation hierarchy, be it not the lowest level? Should, on the other hand, the answer to Problem 1 be positive, we might try to establish even stronger results such as a positive answer to

Problem 3. Is (FO+LFP) $\cap$ EXT $\subseteq$ Datalog($\neq, \neg$)?

6.2 Preservation under Homomorphisms

In this subsection we briefly turn our attention to a different preservation property. A *homomorphism* from A to B is a map $h : A \mapsto B$ such that for all n-ary relation symbols $R(\overline{x})$, and for all n-tuples $\overline{a} \subseteq A$, if $A \models R(\overline{a})$, then $B \models R(h(\overline{a}))$. A class of models $\mathcal{C}$ is closed under homomorphisms iff for all A and B such that there is a homomorphism from A to B, if $A \in \mathcal{C}$, then $B \in \mathcal{C}$. Let HOM denote the set of classes in $\mathcal{F}$ that are closed under homomorphisms. A sentence φ in FO, $L^{\omega}_{\infty\omega}$, etc. is *positive*, if and only if, it does not contain any negations. The following well-known classical result is a direct consequence of the Los-Tarski Theorem: for all $\varphi \in$ FO, Mod(φ) is closed under homomorphisms, if and only if, φ is equivalent to a positive existential sentence. This theorem is one of a few classical results whose validity over $\mathcal{F}$ remains unknown. In our current notation, we can formulate the question as the following open problem, the interest of which has been emphasized by Gurevich [10] and Kolaitis (see [8]).

Problem 4. Is FO $\cap$ HOM $\subseteq$ FO$^+(\exists)$?

(To avoid confusion, it should be remarked that although [10] announces a solution to Problem 4, this claim has been withdrawn.)

The following proposition yields some information about the homomorphism preservation question. We direct the reader to [17] for its proof.

Proposition 31. Datalog($\neq, \neg$) $\cap$ HOM $\subseteq$ Datalog.

Propositions 29, 30, and 31 yield as an immediate corollary the following special case of the homomorphism preservation theorem.

Corollary 32. FO[$\exists^*\forall\exists$] $\cap$ HOM $=$ FO$^+(\exists)$.

References

1. F. Afrati, S. Cosmadakis, and M. Yannakakis. On datalog vs. polynomial time. In *Proceedings of the 10th ACM Symposium on Principles of Database Systems*, 1991.
2. M. Ajtai and Y. Gurevich. Datalog vs. first-order logic. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 142–146, 1989.
3. H. Andreka, J. van Benthem, and I. Nemeti. Submodel preservation theorems in finite variable fragments. In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra.* Cambridge University Press, 1994.
4. J. Barwise. On Moschovakis closure ordinals. *Journal of Symbolic Logic*, 42:292–296, 1977.
5. A. Dawar, S. Lindell, and S. Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, to appear.
6. R. Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41(1):50–58, March 1976.
7. H. Gaifman. Concerning measures in first-order calculi. *Israel Journal of Mathematics*, 2:1–18, 1964.
8. E. Grädel and J. Tyszkiewicz, editors. *Problems in Finite Model Theory.* Available via anonymous ftp from ftp.informatik.rwth-aachen.de, version of May 25, 1994.

9. Y. Gurevich. Toward logic tailored for computational complexity. In M. Richter et al., editors, *Computation and Proof Theory*, pages 175–216. Springer Lecture Notes in Mathematics, 1984.
10. Y. Gurevich. On finite model theory (extended abstract). In S. R. Buss and P. J. Scott, editors, *Feasible Mathematics*, pages 211–219. Birkhauser, 1990.
11. N. Immerman. Upper and lower bounds for first-order expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.
12. Ph. G. Kolaitis and M. Y. Vardi. On the expressive power of datalog: tools and a case study. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 61–71, 1990.
13. Ph. G. Kolaitis and M. Y. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *Proceedings of the 7th IEEE Symposium on Logic in Computer Science*, pages 46–57, 1992.
14. Ph. G. Kolaitis and M. Y. Vardi. Infinitary logics and 0-1 laws. *Information and Computation*, 98(2):258–294, 1992.
15. Libo Lo. Preservation theorems of finite models (abstract). *Journal of Symbolic Logic*, 58:376, 1993.
16. B. Poizat. Deux ou trois choses que je sais de L_n. *Journal of Symbolic Logic*, 47(3):641–658, 1982.
17. E. Rosen. Finite model theory and finite variable logics, 1995. In preparation.
18. W. Tait. A counterexample to a conjecture of Scott and Suppes. *Journal of Symbolic Logic*, 24(1):15–16, 1959.

A Query Language for NC (Extended Abstract)

Dan Suciu and Val Breazu-Tannen*

University of Pennsylvania
email: [suciu,val]@cis.upenn.edu

Abstract. We show that a form of divide and conquer recursion on sets together with the relational algebra expresses exactly the queries over ordered relational databases which are NC-computable. At a finer level, we relate k nested uses of recursion exactly to AC^k, $k \geq 1$. We also give corresponding results for complex objects.[2]

1 Introduction

NC is the complexity class of functions that are computable in polylogarithmic time with polynomially many processors on a *parallel random access machine* (PRAM). The query language for NC discussed here is centered around a *divide and conquer recursion* (*dcr*) on sets which has obvious potential for parallel evaluation and can easily express, for example, transitive closure and parity. *dcr* with parameters e, f, u defines the unique function $\varphi = dcr(e, f, u)$ such that:

$$\begin{aligned}\varphi(\emptyset) &\overset{\text{def}}{=} e\\ \varphi(\{y\}) &\overset{\text{def}}{=} f(y)\\ \varphi(s_1 \cup s_2) &\overset{\text{def}}{=} u(\varphi(s_1), \varphi(s_2)) \text{ when } s_1 \cap s_2 = \emptyset\end{aligned}$$

For parity, we take $e \overset{\text{def}}{=} \mathit{false}$, $f(y) \overset{\text{def}}{=} \mathit{true}$ and $u(v_1, v_2) \overset{\text{def}}{=} v_1 \ \mathit{xor} \ v_2$. To compute the transitive closure of some binary relation r, take $e \overset{\text{def}}{=} \emptyset$, $f(y) \overset{\text{def}}{=} r$ and $u(r_1, r_2) \overset{\text{def}}{=} r_1 \cup r_2 \cup r_1 \circ r_2$. Then, the transitive closure of r is obtained by applying φ to the set of nodes of the relation r, namely $tc(r) = \varphi(\Pi_1(r) \cup \Pi_2(r))$, where Π_1, Π_2 are the relational projections. In general, $dcr(e, f, u)$ is well-defined when there is some set containing e and the range of f, on which u is associative, commutative and has the identity e. For parity, this is the set $\mathbb{B}$ of booleans, while for transitive closure, it is the set $\{r \cup r^2 \cup \ldots \cup r^n \mid n \geq 0\}$.

* The authors were partially supported by NSF Grant CCR-90-57570 and ONR Contract NOOO14-93-11284

[2] These results were first announced at PODS'94 [30] whose proceedings contain a somewhat more detailed exposition than the one below. A journal version is in preparation. Here we are giving just enough definitions to clarify the statement of the main results. See also http://www.cis.upenn.edu/~wfan/dbl.html

We show that *dcr* together with the relational algebra expresses exactly the queries over *ordered* databases of flat relations that are NC-computable. We also show that a bounded version of *dcr* together with the nested relational algebra expresses exactly the queries over *ordered* databases of complex objects that are NC-computable. In fact, we prove the more refined versions that relate k nested uses of (bounded) *dcr* exactly to the subclass AC^k of NC where $k \geq 1$. Some explanations are in order:

- Computable queries are in the sense of Chandra and Harel [10], naturally extended to complex objects.

- Any language that can express the class of queries expressed by first-order logic would do just as well as the relational algebra. Similarly for complex objects, where a corresponding class of tractable queries has emerged from several equivalent formalisms. Some of these formalisms are syntactically restricted higher-order logics, others are algebraic languages, often called nested relational algebras, hence our phrasing above.

- *dcr* and (nested) relational algebra have meaning over any (nested) relational database. But, as with all known characterizations of query complexity classes below NP, we know how to capture the entire NC only over ordered databases. Formally, we do this by extending the language with an order predicate.

- A bounded version of *dcr* is necessary over complex objects, otherwise queries of high complexity such as *powerset* will be expressible. The bounded version is obtained by intersecting the result with a bounding set at each recursion step (section 2). This is similar to the bounded fixpoints studied in [29], and, as with fixpoints, over flat relations *dcr* can always be expressed through bounded *dcr* (section 2).

We believe that these results are of interest from two angles.

∠ **Query language design.** *dcr* is a well-known construct. It appears under the name *pump*, in a language specifically designed for a parallel database machine, FAD [4]. Following FAD, but under the name *hom*, it was included in Machiavelli [24] where it fit nicely into the language's type system. Called (a form of) *transducer*, it is part of SVP [27], precisely in order to support divide and conquer parallelism. Some limitations of its theoretical expressive power were examined (under the name *hom*) by Immerman, Patnaik, and Stemple ([22] theorem 7.8). They also note that *dcr* is in NC.

As part of a larger group of researchers, we became interested in *dcr* because it fits into a natural hierarchy of query languages that share a common semantic basis, built around forms of structural recursion on collection types [7, 6, 8] (see section 2). Theoretical studies of expressiveness, such as [33, 6, 29] and the present paper help us with the choice and mix of primitives, as well as implementation strategies. In particular, *dcr* is at the core of a sublanguage for which we are currently seeking efficient implementation techniques for a variety of parallel architectures.

∠ **Computational complexity.** Following Vardi and Immerman's influential result [32, 18] that first-order logic with least fixed point captures exactly the *PTIME*-computable queries on flat relations over ordered databases, several characterizations of low complexity classes in terms of logics or algebras used in databases have been discovered with the hope that logical methods may give insights into the difficult problem of complexity class separation. We mention first a few of these characterizations which have had a direct influence on the work here.

For parallel complexity classes, Immerman [21] shows that the class of finite and ordered relational structures recognizable in parallel time $t(n)$ (n is the size of the structure) on a certain CRCW (concurrent read - concurrent write) PRAM coincides with the class of structures definable by a first-order induction [23] of depth up to $t(n)$. For complex object databases, Grumbach and Vianu [16, 15] give a syntactic restriction of the ramified higher-order logic CALC which, together with inflationary fixpoints and in the presence of order, captures exactly the *PTIME*-computable complex-object queries. Suciu [29] shows that, in the presence of order, the same class of queries is captured by the nested relational algebra augmented with an inflationary bounded fixpoint operator.

To the best of our knowledge, no characterization of parallel complexity classes of queries over complex objects has been given before. What is more likely to set our results apart, however, is the *intrinsic* nature of the language we are proposing: the semantics of *dcr* puts it naturally in *NC*; there is no need to impose logarithmic bounds on the number of iterations or recursion depth. Moreover, it can be shown that a different kind of recursion on sets, namely structural recursion on the insert presentation of sets ([7]; notation *sri*; definitions reviewed in section 2), together with the relational algebra expresses exactly the *PTIME*-computable queries on ordered databases. This follows from results in [22]; we state the corresponding result for complex objects in proposition 8. Hence, at least over ordered databases, the difference between *NC* and *PTIME* boils down to two different ways of recurring on sets, divide and conquer vs. element by element.

Gurevich [17] and Compton and Laflamme [12] characterize the *DLOGSPACE*- and respectively the NC^1-computable global functions on finite and ordered relational structures as algebras with certain primitive recursion schemas. Compton and Laflamme capture NC^1 also with first-order logic augmented with *BIT*. and with an operator for *defining relations by primitive recursion.* The kinds of recursions used in these two papers are very different from *dcr* because they depend on some linear ordering of the underlying structures for their actual definition. While *dcr* is a form of recursion on finite sets, these recursions are on notations for elements of (linearly ordered) finite sets. Of course, we do not attempt to characterize *DLOGSPACE* or NC^1 or, for that matter, any class below AC^1, but see Immerman's characterizations of such classes in terms of languages more in the spirit of ours than of those of Gurevich, Compton, and Laflamme [20, 19].

We should also mention here the work of Clote [11] for related characterizations of most parallel complexity classes of *arithmetical* functions.

We should point out, however, one sense in which our language is not as neat as, say, first-order logic with least fixpoint, which captures *PTIME* in the presence of order, or first-order logic with transitive closure, which captures *NLOGSPACE* in the presence of order. For *dcr* to be well-defined, the operations involved in it must satisfy certain algebraic identities (associativity, commutativity, identity) and this turns out to be an undecidable condition (in fact Π_1^0 complete; see section 2). Of course, only a certain family of instances of *dcr* is actually needed in the simulations, and for these, the algebraic conditions always hold. Hence, it is of theoretical interest that there is a decidable sublanguage of *dcr* plus relational algebra which captures exactly *NC* in the presence of order. In practice, we have found it useful to provide special syntax for some instances of *dcr* in which the algebraic conditions are automatically satisfied, but we found it counterproductive to limit *dcr* to these instances, as other uses kept appearing.

2 Recursion on Sets

A fruitful approach to the design of query languages for complex objects is to consider tuples and sets as orthogonal [6, 8]. Hence, there will be primitives that work on tuples, primitives that work on sets, and general primitives for combining other primitives. In this section we discuss several choices for defining recursions on sets and the relationship between them. One of these forms of yrecursion—*dcr*—will be at the core of our "query languages for *NC*". A precise definition of the languages will be given in section 3.

Complex objects are built essentially from tuples and finite sets. To describe them, we define the **complex object types** by the grammar:

$$t \overset{\text{def}}{=} \mathbb{D} \mid \mathbb{B} \mid unit \mid t \times t \mid \{t\}$$

where $\mathbb{D}$ is some base type, $\mathbb{B}$ is the type of booleans and *unit* is the type containing only the empty tuple ($unit = \{()\}$). The values of type $s \times t$ are pairs (x, y) with $x \in s, y \in t$, and the values of type $\{t\}$ are finite sets of elements from t. Products of types of the form $\{s\}$, with s a product of base types ($\mathbb{D}, \mathbb{B}$, *unit*), are called **flat types**. We will also need to consider **function types** having the form $s \rightarrow t$, where s and t are complex object types.

There seem to be two basic ways of describing the structure of finite sets. In one way, they are generated by finitely many (maybe zero!) binary unions of singleton sets. We call this the *union presentation.* In another way, they are generated by finitely many insertions of one element, starting with the empty set. We call this the *insert presentation.* Recognizing the relevant algebraic identities satisfied by union (associativity, commutativity, idempotence, has $\emptyset$ as an identity) and by element insertion (i-commutativity and i-idempotence) gives us two different algebraic structures on finite sets. Both these algebras are characterized by universality properties, which amount to definitions of functions by *structural recursion* [7, 6]. We have a structural recursion construct on the union presentation, *sru*:

$$\frac{e : t \quad f : s \rightarrow t \quad u : t \times t \rightarrow t}{sru(e, f, u) : \{s\} \rightarrow t}$$

$$\begin{aligned}
sru(e,f,u)(\emptyset) &\overset{\text{def}}{=} e \\
sru(e,f,u)(\{y\}) &\overset{\text{def}}{=} f(y) \\
sru(e,f,u)(s_1 \cup s_2) &\overset{\text{def}}{=} u(sru(e,f,u)(s_1), \\
&\qquad sru(e,f,u)(s_2))
\end{aligned}$$

$sru(e,f,u)$ is well-defined when there is some subset of t containing e and the range of f, on which u is associative, commutative, idempotent, and has the identity e. We also have a structural recursion construct on the insert presentation, $sri(x \upharpoonright s)$ is the element insertion operation, $\{x\} \cup s$):

$$\frac{e : t \quad i : s \times t \to t}{sri(e,i) : \{s\} \to t}$$

$$\begin{aligned}
sri(e,i)(\emptyset) &\overset{\text{def}}{=} e \\
sri(e,i)(y \upharpoonright s) &\overset{\text{def}}{=} i(y, sri(e,i)(s))
\end{aligned}$$

$sri(e,i)$ is well-defined when i is *i-commutative*, and *i-idempotent* ($i(x, i(y,s)) = i(y, i(x,s))$, $i(x, i(x,s)) = i(x,s)$) on some subset of t containing e.

The *divide and conquer recursion* dcr, whose definition was already given in section 1, is superficially related to sru. For example $sru(e,f,u)$ is well-defined then so is $dcr(e,f,u)$ and they are equal. But dcr is potentially more expressive, since u need not be idempotent. In fact, we do not know if sru can express parity or transitive closure. One can prove:

Proposition 1. *sri can express dcr, which in turn can express sru [7, 30]. All these are done with at most polynomial overhead.*

Over complex objects dcr can express *powerset* hence we need some restriction if we are to stay within NC. An analog to Peter Buneman's idea of *bounded* fixpoints [29] does the job. We define a **PS-type** (product of sets) to be either a set type, or a product of PS-types. The **bounded** version of dcr is defined by:

$$\frac{e : t \quad f : s \to t \quad u : t \times t \to t \quad b : t}{bdcr(e,f,u,b) : \{s\} \to t}$$

where t is a PS-type, with the semantics:

$$bdcr(e,f,u,b) \overset{\text{def}}{=} dcr(e \cap b, f \cap b, u \cap b)$$

(here $(u \cap b)(y,y') \overset{\text{def}}{=} u(y,y') \cap b$, etc). As for dcr, we define the **bounded** sri, $bsri(e,i,b)$, for some PS-type t, to be $sri(e \cap b, i \cap b)$. Proposition 1 easily extends to the bounded versions of recursion. Over flat relations the explicit bounding is unnecessary:

Proposition 2. *bdcr together with the relational algebra can express dcr when its arguments are flat relations and its values are of flat PS-type. Similarly for bsri and sri.*

Unfortunately, for a language at least as expressive as first-order logic, verifying most of the identities required by the *sri*, *dcr*, and *sru* constructs is as hard as testing the validity of a first-order formula in all finite models, hence it is a Π_1^0-complete question.

3 Query Languages

In this section we define the languages that offer characterizations of NC over complex objects and over flat relations. For complex objects, the language is obtained by adding *bdcr* and an order predicate to the *nested relational algebra* $\mathcal{NRA}$. This name can be used to describe a language that has the same expressive power as Schek and Scholl's NF^2 relational algebra [28], or Thomas and Fischer's algebra [31], or Paredaens and Van Gucht's *nested algebra* [25, 26], or Abiteboul and Beeri's *algebra without powerset* [2]. The particular formalization we use here was essentially introduced in [8] and is based on what is called there the **monad calculus**.

For each type s we assume an infinite set of variables to be given, and write x^s for a variable of type s. As usual, we distinguish between free and bound variables, and we identify those expressions that differ only in the name of some bound variables. $\mathcal{NRA}$ is defined by the rules in figure 1.

$$\frac{}{x^t : t} \qquad \frac{e_1 : t_1 \quad e_2 : t_2}{(e_1, e_2) : (t_1, t_2)}$$

$$\frac{e : t_1 \times t_2}{\pi_1(e) : t_1} \qquad \frac{e : t_1 \times t_2}{\pi_2(e) : t_2}$$

$$\frac{}{\emptyset : \{t\}} \qquad \frac{e : t}{\{e\} : \{t\}} \qquad \frac{e_1 : \{t\} \quad e_1 : \{t\}}{e_1 \cup e_2 : \{t\}}$$

$$\frac{e_1 : \mathbf{D} \quad e_2 : \mathbf{D}}{e_1 = e_2 : \mathbf{B}} \qquad \frac{}{() : unit}$$

$$\frac{e : \{t\}}{empty(e) : \mathbf{B}} \qquad \frac{e : \mathbf{B} \quad e_1 : t \quad e_2 : t}{if\ e\ then\ e_1\ else\ e_2 : t}$$

$$\frac{e : t}{\lambda x^s.e : s \to t} \qquad \frac{f : s \to t \quad e : s}{f(e) : t}$$

$$\frac{f : s \to \{t\}}{ext(f) : \{s\} \to \{t\}}$$

Fig. 1. The Nested Relational Algebra

We briefly describe the semantics of the expressions: (e_1, e_2) constructs a tuple, π_1, π_2 are the projections, $\{e\}$ is the singleton set, $empty(e)$ returns true iff $e = \emptyset$, *if* e *then* e_1 *else* e_2 equals e_1 iff $e =$ *true* and e_2 otherwise, $\lambda x^s.e$ denotes a function whose input is the variable x^s, $f(e)$ is function application, and $ext(f)(\{x_1, \ldots, x_n\}) \overset{\text{def}}{=} f(x_1) \cup \ldots \cup f(x_n)$. A possible set Σ of **external functions** $p : dom(p) \rightarrow codom(p)$ could be added to the language; in this case, we denote the language by $\mathcal{NRA}(\Sigma)$. We abbreviate with $\lambda(x, y).e$ the expression $\lambda z.e[\pi_1(z)/x, \pi_2(z)/y]$.

$\mathcal{NRA}$ is powerful enough to express the following functions: set difference, set intersection, cartesian product, database projections, equalities at all types, selections over predicates definable in the language, nest and unnest [8].

We will make two extensions to $\mathcal{NRA}$ in order to obtain a language that characterizes *NC*. One extension is the addition of divide-and-conquer recursion on sets. *dcr* is too powerful on complex objects since it can express *powerset* (actually one can show that $\mathcal{NRA}(dcr)$ has the same expressive power as Abiteboul and Beeri's *algebra* [2]). So instead, we will add the bounded version of this kind of recursion and consider the language $\mathcal{NRA}(bdcr)$. We shall have occasion to contrast this languages with $\mathcal{NRA}(bsri)$. Note that proposition 1 states that $\mathcal{NRA}(bdcr) \subseteq \mathcal{NRA}(bsri)$, and this holds even in the presence of external functions.

The second extension concerns *ordering.* All known characterizations of complexity classes below *PTIME* are about ordered databases, and ours is no exception. We isolate the use of ordering into the addition of an external function $\leq: \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{B}$, which is understood always to denote a linear order on $\mathbb{D}$. The order relation can be lifted to all types.

We have thus reached the language $\mathcal{NRA}(bdcr, \leq)$ which is the subject of one of our main theorems, characterizing the *NC*-computable queries on ordered databases of complex objects (theorem 3).

In the same theorem, we obtain a finer characterization, involving the *AC*-hierarchy, for which we need the notion of **the depth of recursion nesting** $depth(e)$, of some expression e. We define this to be to be the maximum depth of recursions occurring in e. For example, for *bdcr*: $depth(bdcr(e, f, u)) \overset{\text{def}}{=} \max(depth(e), depth(f), 1 + depth(u))$ (only u is actually iterated). We denote $\mathcal{NRA}(bdcr^{(k)})$ the restrictions of the language $\mathcal{NRA}(bdcr)$ to iteration depth $\leq k$. We have thus obtained the hierarchy of languages $\mathcal{NRA}(bdcr^{(k)}, \leq)$ which is the subject of the finer characterization given in theorem 3.

To characterize the *NC*-computable queries over ordered flat relations (theorem 4), we denote by $\mathcal{NRA}^1$ the restriction of $\mathcal{NRA}$ to types of "set height ≤ 1", i.e. the only types allowed in $\mathcal{NRA}^1$ as inputs, outputs and intermediate types are products of base and flat types. Its expressive power is essentially the same as that of the relational algebra [6], hence first-order logic. We make the same two extensions as for $\mathcal{NRA}$, except that here it is safe to use *dcr*. For good measure, proposition 2 states that it doesn't matter: $\mathcal{NRA}^1(bdcr) = \mathcal{NRA}^1(dcr)$. Note however that this fails in the presence of certain external functions. Again we will have occasion to contrast $\mathcal{NRA}^1(dcr)$ with $\mathcal{NRA}^1(sri)$ so note that proposition 1 states that $\mathcal{NRA}^1(dcr) \subseteq \mathcal{NRA}^1(sri)$, and this holds even in the presence of external functions. Hence, the languages of interest for flat relations are $\mathcal{NRA}^1(dcr, \leq)$ and the hierarchy $\mathcal{NRA}^1(dcr^{(k)}, \leq)$.

4 Main Results

Assuming some enumeration of the base type $\mathbb{D}$ to be given, we encode complex objects as strings over the alphabet $A = \{0, 1, \{, \}, (,), comma, blank\}$. The order of elemenets in the sets is irrelevant, but no duplicates are allowed. E.g. the object $\{(a, (b, a)), (c, (a, a)), (a, (b, c))\}$ could be encoded as the string $\{1, (10, 1), (11, (1, 1)), (1, (10, 11))\}$, if the encoding of $\mathbb{D}$ assigns $1, 2, 3$ to a, b, c respectively. Blanks may be arbitrarily scattered in an encoding, but not insided the binary numbers. Furthermore, we make a second encoding of complex objects as binary strings (i.e. in $\{0,1\}^*$) by replacing each of the 8 symbols in A with a 3-bits binary number.

Recall that AC^k is the class of functions $\{0,1\}^* \to \{0,1\}^*$ computable by a "uniform" family of circuits of polynomial size and of depth $O(\log^k n)$ [13]. We define *Q-NC* and *CQ-NC* to be the class of queries over base types and flat relations, and complex objects respectively, which are in *NC*. Similarly, we define the subclasses *Q-AC^k* and *CQ-AC^k*.

Theorem 3. *$\mathcal{NRA}(bdcr, \leq) =$ CQ-NC. More precisely, for every $k \geq 1$ we have $\mathcal{NRA}(bdcr^{(k)}, \leq) =$ CQ-AC^k.*

Theorem 4. *$\mathcal{NRA}^1(dcr, \leq) =$ Q-NC. More precisely, for every $k \geq 1$ we have $\mathcal{NRA}^1(dcr^{(k)}, \leq) =$ Q-AC^k.*

These languages are purely for complex objects, respectively relations. But many external functions of practical interest such as the usual arithmetical operations $(+, *, -, /, etc)$, and the usual aggregate functions (cardinality, sum, average, etc.) are also in *NC*. Can they be added in? The answer is given by:

Proposition 5. *Let Σ be an extension consisting of possible additional base types and a set of functions computable in NC. Then $\mathcal{NRA}(\Sigma, bdcr) \subseteq NC$. However, $\mathcal{NRA}^1(\mathbb{N}, +, dcr)$ can express exponential space queries.*

Immerman in [21] and Barrington, Immerman and Straubing in [5] prove that FO is included in *FO-DCL*-uniform AC^0, and that *FO* together with order and *BIT* relation has the same expressive power as AC^0. Here, we prove that $\mathcal{NRA}$ is included in AC^0, thus extending half of their results to complex objects.

Proposition 6. *All queries in $\mathcal{NRA}(\leq)$, are in FO-DCL-uniform AC^0 (see [5]).*

We state two more results which help us put the main theorems in perspective. Their proofs are omitted from this extended abstract.

Conservative extension. One may wonder in what sense theorem 4 is a "particular case" of theorem 3. Actually, even though the proof of theorem 4 is quite similar to that of theorem 3, theorem 4 in fact follows from theorem 3, proposition 2 and the conservative extension result presented below.

Paredaens and Van Gucht in [26], and Wong in [33] prove that $\mathcal{NRA}$ is a conservative extension of $\mathcal{NRA}^1$. Suciu in [29] proves that $\mathcal{NRA}(bfix)$ is a conservative extension of $\mathcal{NRA}^1(fix)$, where fix is the usual inflationary fixpoint, and $bfix$ is a bounded version of fix. Using the techniques in [29], we can prove the following:

Proposition 7. *Let Σ be a set of external functions which have set heights ≤ 1. Then, $\mathcal{NRA}(\Sigma, bdcr, \leq)$ is a conservative extension of $\mathcal{NRA}^1(\Sigma, bdcr, \leq)$.*

Note that for the case when $\Sigma = \emptyset$, we can turn the tables and proposition 7 follows directly from the main theorems. For the case when $\Sigma \neq 0$, this proposition requires a separate proof, and we are only able to do it in the presence of order. However, we conjecture that $\mathcal{NRA}(bdcr)$ is a conservative extension of $\mathcal{NRA}^1(dcr)$.

PTIME vs. NC. Immerman, Patnaik and Stemple [22] show that *PTIME* is captured by a language built around *set-reduce* (see section 2). Extending their result also to complex objects we have:

Proposition 8. *$\mathcal{NRA}^1(sri, \leq) = \mathcal{NRA}^1(sri^{(1)}, \leq) = PTIME$ [22] and, for complex objects, $\mathcal{NRA}(bsri, \leq) = \mathcal{NRA}(bsri^{(1)}, \leq) = PTIME$.*

Thus, by the main theorems and this proposition, the difference between *PTIME* and *NC* computable queries over ordered databases can be characterized by the difference between two kinds of recursion on sets. It is interesting to note that only one level of recursion nesting suffices for *sri* and *PTIME*, as opposed to *dcr* and *NC*.

5 Conclusions

Ordering seems to play a crucial role in capturing complexity classes below *NP* and our characterization is no exception. Indeed, it follows from theorem 7.8 in [22] that in the absence of ordering $FO + dcr$ cannot express the lower bound in [9] which is in AC^0 plus parity gates ([9] remark 7.2). As with *PTIME*, *DLOGSPACE*, etc., it remains an important open question whether there exists an r.e. set of "programs" that express exactly the *NC*-computable queries over arbitrary relational databases.

On the other hand, studying the expressiveness of the various forms of recursion on sets in the absence of ordering is quite relevant to query language design. It may also be relevant to complexity theory, if an analog to the surprising result of Abiteboul and Vianu [3] holds. They have shown that $PTIME \neq PSPACE$ iff first-order least fixpoint queries $\neq$ first-order *while* queries. (Vardi had shown that in the presence of order the *FO+while* captures *PSPACE* [32].) Dawar, Lindell, and Weinstein [14] give a machine-independent proof of the Abiteboul and Vianu result making use of properties of bounded variable logics. Abiteboul, Vardi and Vianu [1] give evidence for the robustness of the idea with several such results for other pairs of complexity classes. In our case, the analog would be: $NC \neq PTIME$ iff $FO + dcr \neq FO + sri$ (in our formalism, $\mathcal{NRA}^1(dcr) \neq \mathcal{NRA}^1(sri)$). By setting aside the ordering, with its potential for tricky encodings, this would strengthen the observation (section 4) that the difference between tractable sequential and tractable parallel computation can be characterized as the difference between two ways of recurring on sets.

Acknowledgements. We thank Scott Weinstein for many illuminating discussions, Neil Immerman for answering our sometimes naive queries, Peter Buneman and Leonid Libkin for suggestions from a careful reading of an earlier version of this paper, and Peter, Leonid, and Limsoon Wong for their constant help.

References

1. S. Abiteboul, M. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. In *Structure and Complexity*, 1992.
2. Serge Abiteboul and Catriel Beeri. On the power of languages for the manipulation of complex objects. In *Proceedings of International Workshop on Theory and Applications of Nested Relations and Complex Objects*, Darmstadt, 1988. Also available as INRIA Technical Report 846.
3. Serge Abiteboul and Victor Vianu. Generic computation and its complexity. In *Proceedings of 23rd ACM Symposium on the Theory of Computing*, 1991.
4. F. Bancilhon, T. Briggs, S. Khoshafian, and P. Valduriez. A powerful and simple database language. In *Proceedings of 14th International Conference on Very Large Data Bases*, pages 97–105, 1988.
5. David Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.
6. V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece*, pages 9–19. Morgan Kaufmann, August 1991. Also available as UPenn Technical Report MS-CIS-92-17.
7. V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with Sets/Bags/Lists. In *LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.
8. Val Breazu-Tannen, Peter Buneman, and Limsoon Wong. Naturally embedded query languages. In J. Biskup and R. Hull, editors, *LNCS 646: Proceedings of 4th International Conference on Database Theory, Berlin, Germany, October, 1992*, pages 140–154. Springer-Verlag, October 1992. Available as UPenn Technical Report MS-CIS-92-47.
9. Jin-Yi Cai, Martin Furer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
10. Ashok Chandra and David Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.
11. P. Clote. Sequential, machine-independent characterizations of the parallel complexity classes *AlogTime*, AC^k, NC^k, and NC. In Samuel R. Buss and Philip J. Scot, editors, *Feasible Mathematics*. Birkhäuser, Boston, 1990.
12. Kevin L. Compton and Claude Laflamme. An algebra and a logic for NC. *Information and Computation*, 87(1/2):240–262, 1990.

13. S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

14. Anuj Dawar, Steven Lindell, and Scott Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 1993. To appear. Available as UPenn Technical Report MS-CIS-91-97.

15. Stephane Grumbach and Victor Vianu. Expressiveness and complexity of restricted languages for complex objects. In *Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece*, pages 191–202. Morgan Kaufmann, August 1991.

16. Stephane Grumbach and Victor Vianu. Tractable query languages for complex object databases. Technical Report 1573, INRIA, Rocquencourt BP 105, 78153 Le Chesnay, France, December 1991. Extended abstract appeared in PODS 91.

17. Y. Gurevich. Algebra of feasible functions. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, 1983.

18. Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

19. Neil Immerman. Expressibility as a complexity measure: Results and directions. In *Proceedings of 2nd Conference on Structure in Complexity Theory*, pages 194–202, 1987.

20. Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

21. Neil Immerman. Expressibility and parallel complexity. *SIAM Journal of Computing*, 18:625–638, 1989.

22. Neil Immerman, Sushant Patnaik, and David Stemple. The expressiveness of a family of finite set languages. In *Proceedings of 10th ACM Symposium on Principles of Database Systems*, pages 37–52, 1991.

23. Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974.

24. A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli, a polymorphic language with static type inference. In James Clifford, Bruce Lindsay, and David Maier, editors, *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 46–57, Portland, Oregon, June 1989.

25. Jan Paredaens and Dirk Van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. In *Proceedings of 7th ACM Symposium on Principles of Database Systems,Austin, Texas*, pages 29–38, 1988.

26. Jan Paredaens and Dirk Van Gucht. Converting nested relational algebra expressions into flat algebra expressions. *ACM Transaction on Database Systems*, 17(1):65–93, March 1992.

27. D. Stott Parker, Eric Simon, and Patrick Valduriez. SVP: A model capturing sets, streams, and parallelism. In Li-Yan Yuan, editor, *Proceedings of 18th International Conference on Very Large Databases, Vancouver, August 1992*, pages 115–126, San Mateo, California, August 1992. Morgan-Kaufmann.

28. H.-J. Schek and M. H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

29. Dan Suciu. Fixpoints and bounded fixpoints for complex objects. In Catriel Beeri, Atsushi Ohori, and Dennis Shasha, editors, *Proceedings of 4th International Workshop on Database Programming Languages, New York, August 1993*, pages 263–281. Springer-Verlag, January 1994. See also UPenn Technical Report MS-CIS-93-32.

30. Dan Suciu and Val Breazu-Tannen. A query language for NC. In *Proceedings of 13th ACM Symposium on Principles of Database Systems*, pages 167–178, Minneapolis, Minnesota, May 1994. See also UPenn Technical Report MS-CIS-94-05.
31. S. J. Thomas and P. C. Fischer. Nested relational structures. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research: The Theory of Databases*, pages 269–307, London, England, 1986. JAI Press.
32. M. Y. Vardi. The complexity of relational query languages. In *Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing*, pages 137–146, San Francisco, California, 1982.
33. Limsoon Wong. Normal forms and conservative properties for query languages over collection types. In *Proceedings of 12th ACM Symposium on Principles of Database Systems*, pages 26–36, Washington, D. C., May 1993. See also UPenn Technical Report MS-CIS-92-59.

Springer-Verlag and the Environment

We at Springer-Verlag firmly believe that an international science publisher has a special obligation to the environment, and our corporate policies consistently reflect this conviction.

We also expect our business partners – paper mills, printers, packaging manufacturers, etc. – to commit themselves to using environmentally friendly materials and production processes.

The paper in this book is made from low- or no-chlorine pulp and is acid free, in conformance with international standards for paper permanency.

Lecture Notes in Computer Science

For information about Vols. 1–886

please contact your bookseller or Springer-Verlag

Vol. 887: M. Toussaint (Ed.), Ada in Europe. Proceedings, 1994. XII, 521 pages. 1994.

Vol. 888: S. A. Andersson (Ed.), Analysis of Dynamical and Cognitive Systems. Proceedings, 1993. VII, 260 pages. 1995.

Vol. 889: H. P. Lubich, Towards a CSCW Framework for Scientific Cooperation in Europe. X, 268 pages. 1995.

Vol. 890: M. J. Wooldridge, N. R. Jennings (Eds.), Intelligent Agents. Proceedings, 1994. VIII, 407 pages. 1995. (Subseries LNAI).

Vol. 891: C. Lewerentz, T. Lindner (Eds.), Formal Development of Reactive Systems. XI, 394 pages. 1995.

Vol. 892: K. Pingali, U. Banerjee, D. Gelernter, A. Nicolau, D. Padua (Eds.), Languages and Compilers for Parallel Computing. Proceedings, 1994. XI, 496 pages. 1995.

Vol. 893: G. Gottlob, M. Y. Vardi (Eds.), Database Theory – ICDT '95. Proceedings, 1995. XI, 454 pages. 1995.

Vol. 894: R. Tamassia, I. G. Tollis (Eds.), Graph Drawing. Proceedings, 1994. X, 471 pages. 1995.

Vol. 895: R. L. Ibrahim (Ed.), Software Engineering Education. Proceedings, 1995. XII, 449 pages. 1995.

Vol. 896: R. N. Taylor, J. Coutaz (Eds.), Software Engineering and Human-Computer Interaction. Proceedings, 1994. X, 281 pages. 1995.

Vol. 897: M. Fisher, R. Owens (Eds.), Executable Modal and Temporal Logics. Proceedings, 1993. VII, 180 pages. 1995. (Subseries LNAI).

Vol. 898: P. Steffens (Ed.), Machine Translation and the Lexicon. Proceedings, 1993. X, 251 pages. 1995. (Subseries LNAI).

Vol. 899: W. Banzhaf, F. H. Eeckman (Eds.), Evolution and Biocomputation. VII, 277 pages. 1995.

Vol. 900: E. W. Mayr, C. Puech (Eds.), STACS 95. Proceedings, 1995. XIII, 654 pages. 1995.

Vol. 901: R. Kumar, T. Kropf (Eds.), Theorem Provers in Circuit Design. Proceedings, 1994. VIII, 303 pages. 1995.

Vol. 902: M. Dezani-Ciancaglini, G. Plotkin (Eds.), Typed Lambda Calculi and Applications. Proceedings, 1995. VIII, 443 pages. 1995.

Vol. 903: E. W. Mayr, G. Schmidt, G. Tinhofer (Eds.), Graph-Theoretic Concepts in Computer Science. Proceedings, 1994. IX, 414 pages. 1995.

Vol. 904: P. Vitányi (Ed.), Computational Learning Theory. EuroCOLT'95. Proceedings, 1995. XVII, 415 pages. 1995. (Subseries LNAI).

Vol. 905: N. Ayache (Ed.), Computer Vision, Virtual Reality and Robotics in Medicine. Proceedings, 1995. XIV,

Vol. 906: E. Astesiano, G. Reggio, A. Tarlecki (Eds.), Recent Trends in Data Type Specification. Proceedings, 1995. VIII, 523 pages. 1995.

Vol. 907: T. Ito, A. Yonezawa (Eds.), Theory and Practice of Parallel Programming. Proceedings, 1995. VIII, 485 pages. 1995.

Vol. 908: J. R. Rao Extensions of the UNITY Methodology: Compositionality, Fairness and Probability in Parallelism. XI, 178 pages. 1995.

Vol. 909: H. Comon, J.-P. Jouannaud (Eds.), Term Rewriting. Proceedings, 1993. VIII, 221 pages. 1995.

Vol. 910: A. Podelski (Ed.), Constraint Programming: Basics and Trends. Proceedings, 1995. XI, 315 pages. 1995.

Vol. 911: R. Baeza-Yates, E. Goles, P. V. Poblete (Eds.), LATIN '95: Theoretical Informatics. Proceedings, 1995. IX, 525 pages. 1995.

Vol. 912: N. Lavrac, S. Wrobel (Eds.), Machine Learning: ECML – 95. Proceedings, 1995. XI, 370 pages. 1995. (Subseries LNAI).

Vol. 913: W. Schäfer (Ed.), Software Process Technology. Proceedings, 1995. IX, 261 pages. 1995.

Vol. 914: J. Hsiang (Ed.), Rewriting Techniques and Applications. Proceedings, 1995. XII, 473 pages. 1995.

Vol. 915: P. D. Mosses, M. Nielsen, M. I. Schwartzbach (Eds.), TAPSOFT '95: Theory and Practice of Software Development. Proceedings, 1995. XV, 810 pages. 1995.

Vol. 916: N. R. Adam, B. K. Bhargava, Y. Yesha (Eds.), Digital Libraries. Proceedings, 1994. XIII, 321 pages. 1995.

Vol. 917: J. Pieprzyk, R. Safavi-Naini (Eds.), Advances in Cryptology - ASIACRYPT '94. Proceedings, 1994. XII, 431 pages. 1995.

Vol. 918: P. Baumgartner, R. Hähnle, J. Posegga (Eds.), Theorem Proving with Analytic Tableaux and Related Methods. Proceedings, 1995. X, 352 pages. 1995. (Subseries LNAI).

Vol. 919: B. Hertzberger, G. Serazzi (Eds.), High-Performance Computing and Networking. Proceedings, 1995. XXIV, 957 pages. 1995.

Vol. 920: E. Balas, J. Clausen (Eds.), Integer Programming and Combinatorial Optimization. Proceedings, 1995. IX, 436 pages. 1995.

Vol. 921: L. C. Guillou, J.-J. Quisquater (Eds.), Advances in Cryptology – EUROCRYPT '95. Proceedings, 1995. XIV, 417 pages. 1995.

Vol. 922: H. Dörr, Efficient Graph Rewriting and Its Implementation. IX, 266 pages. 1995.

Vol. 923: M. Meyer (Ed.), Constraint Processing. IV, 289 pages. 1995.

Vol. 924: P. Ciancarini, O. Nierstrasz, A. Yonezawa (Eds.), Object-Based Models and Languages for Concurrent Systems. Proceedings, 1994. VII, 193 pages. 1995.

Vol. 925: J. Jeuring, E. Meijer (Eds.), Advanced Functional Programming. Proceedings, 1995. VII, 331 pages. 1995.

Vol. 926: P. Nesi (Ed.), Objective Software Quality. Proceedings, 1995. VIII, 249 pages. 1995.

Vol. 927: J. Dix, L. Moniz Pereira, T. C. Przymusinski (Eds.), Non-Monotonic Extensions of Logic Programming. Proceedings, 1994. IX, 229 pages. 1995. (Subseries LNAI).

Vol. 928: V.W. Marek, A. Nerode, M. Truszczynski (Eds.), Logic Programming and Nonmonotonic Reasoning. Proceedings, 1995. VIII, 417 pages. 1995. (Subseries LNAI).

Vol. 929: F. Morán, A. Moreno, J.J. Merelo, P. Chacón (Eds.), Advances in Artificial Life. Proceedings, 1995. XIII, 960 pages. 1995 (Subseries LNAI).

Vol. 930: J. Mira, F. Sandoval (Eds.), From Natural to Artificial Neural Computation. Proceedings, 1995. XVIII, 1150 pages. 1995.

Vol. 931: P.J. Braspenning, F. Thuijsman, A.J.M.M. Weijters (Eds.), Artificial Neural Networks. IX, 295 pages. 1995.

Vol. 932: J. Iivari, K. Lyytinen, M. Rossi (Eds.), Advanced Information Systems Engineering. Proceedings, 1995. XI, 388 pages. 1995.

Vol. 933: L. Pacholski, J. Tiuryn (Eds.), Computer Science Logic. Proceedings, 1994. IX, 543 pages. 1995.

Vol. 934: P. Barahona, M. Stefanelli, J. Wyatt (Eds.), Artificial Intelligence in Medicine. Proceedings, 1995. XI, 449 pages. 1995. (Subseries LNAI).

Vol. 935: G. De Michelis, M. Diaz (Eds.), Application and Theory of Petri Nets 1995. Proceedings, 1995. VIII, 511 pages. 1995.

Vol. 936: V.S. Alagar, M. Nivat (Eds.), Algebraic Methodology and Software Technology. Proceedings, 1995. XIV, 591 pages. 1995.

Vol. 937: Z. Galil, E. Ukkonen (Eds.), Combinatorial Pattern Matching. Proceedings, 1995. VIII, 409 pages. 1995.

Vol. 938: K.P. Birman, F. Mattern, A. Schiper (Eds.), Theory and Practice in Distributed Systems. Proceedings,1994. X, 263 pages. 1995.

Vol. 939: P. Wolper (Ed.), Computer Aided Verification. Proceedings, 1995. X, 451 pages. 1995.

Vol. 940: C. Goble, J. Keane (Eds.), Advances in Databases. Proceedings, 1995. X, 277 pages. 1995.

Vol. 941: M. Cadoli, Tractable Reasoning in Artificial Intelligence. XVII, 247 pages. 1995. (Subseries LNAI).

Vol. 942: G. Böckle, Exploitation of Fine-Grain Parallelism. IX, 188 pages. 1995.

Vol. 943: W. Klas, M. Schrefl, Metaclasses and Their Application. IX, 201 pages. 1995.

Vol. 944: Z. Fülöp, F. Gécseg (Eds.), Automata, Languages and Programming. Proceedings, 1995. XIII, 686 pages. 1995.

Vol. 945: B. Bouchon-Meunier, R.R. Yager, L.A. Zadeh (Eds.), Advances in Intelligent Computing - IPMU '94. Proceedings, 1994. XII, 628 pages.1995.

Vol. 946: C. Froidevaux, J. Kohlas (Eds.), Symbolic and Quantitative Approaches to Reasoning and Uncertainty. Proceedings, 1995. X, 420 pages. 1995. (Subseries LNAI).

Vol. 947: B. Möller (Ed.), Mathematics of Program Construction. Proceedings, 1995. VIII, 472 pages. 1995.

Vol. 948: G. Cohen, M. Giusti, T. Mora (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Proceedings, 1995. XI, 485 pages. 1995.

Vol. 949: D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing. Proceedings, 1995. VIII, 361 pages. 1995.

Vol. 950: A. De Santis (Ed.), Advances in Cryptology - EUROCRYPT '94. Proceedings, 1994. XIII, 473 pages. 1995.

Vol. 951: M.J. Egenhofer, J.R. Herring (Eds.), Advances in Spatial Databases. Proceedings, 1995. XI, 405 pages. 1995.

Vol. 952: W. Olthoff (Ed.), ECOOP '95 - Object-Oriented Programming. Proceedings, 1995. XI, 471 pages. 1995.

Vol. 953: D. Pitt, D.E. Rydeheard, P. Johnstone (Eds.), Category Theory and Computer Science. Proceedings, 1995. VII, 252 pages. 1995.

Vol. 954: G. Ellis, R. Levinson, W. Rich. J.F. Sowa (Eds.), Conceptual Structures: Applications, Implementation and Theory. Proceedings, 1995. IX, 353 pages. 1995. (Subseries LNAI).

VOL. 955: S.G. Akl, F. Dehne, J.-R. Sack, N. Santoro (Eds.), Algorithms and Data Structures. Proceedings, 1995. IX, 519 pages. 1995.

Vol. 956: X. Yao (Ed.), Progress in Evolutionary Computation. Proceedings, 1993, 1994. VIII, 314 pages. 1995. (Subseries LNAI).

Vol. 957: C. Castelfranchi, J.-P. Müller (Eds.), From Reaction to Cognition. Proceedings, 1993. VI, 252 pages. 1995. (Subseries LNAI).

Vol. 958: J. Calmet, J.A. Campbell (Eds.), Integrating Symbolic Mathematical Computation and Artificial Intelligence. Proceedings, 1994. X, 275 pages. 1995.

Vol. 959: D.-Z. Du, M. Li (Eds.), Computing and Combinatorics. Proceedings, 1995. XIII, 654 pages. 1995.

Vol. 960: D. Leivant (Ed.), Logic and Computational Complexity. Proceedings, 1994. VIII, 514 pages. 1995.

Vol. 961: K.P. Jantke, S. Lange (Eds.), Algorithmic Learning for Knowledge-Based Systems. X, 511 pages. 1995. (Subseries LNAI).

Vol. 962: I. Lee, S.A. Smolka (Eds.), CONCUR '95: Concurrency Theory. Proceedings, 1995. X, 547 pages. 1995.

Vol. 963: D. Coppersmith (Ed.), Advances in Cryptology —CRYPTO '95. Proceedings, 1995. XII, 467 pages. 1995.

Lecture Notes in Computer Science

This series reports new developments in computer science research and teaching, quickly, informally, and at a high level. The timeliness of a manuscript is more important than its form, which may be unfinished or tentative. The type of material considered for publication includes

- drafts of original papers or monographs,
- technical reports of high quality and broad interest,
- advanced-level lectures.
- reports of meetings, provided they are of exceptional interest and focused on a single topic.

Publication of Lecture Notes is intended as a service to the computer science community in that the publisher Springer-Verlag offers global distribution of documents which would otherwise have a restricted readership. Once published and copyrighted they can be cited in the scientific literature.

Manuscripts

Lecture Notes are printed by photo-offset from the master copy delivered in camera-ready form. Manuscripts should be no less than 100 and preferably no more than 500 pages of text. Authors of monographs and editors of proceedings volumes receive 50 free copies of their book. Manuscripts should be printed with a laser or other high-resolution printer onto white paper of reasonable quality. To ensure that the final photo-reduced pages are easily readable, please use one of the following formats:

Font size	Printing area		Final size
(points)	(cm)	(inches)	(%)
10	12.2 x 19.3	4.8 x 7.6	100
12	15.3 x 24.2	6.0 x 9.5	80

On request the publisher will supply a leaflet with more detailed technical instructions or a TEX macro package for the preparation of manuscripts.

Manuscripts should be sent to one of the series editors or directly to:

Springer-Verlag, Computer Science Editorial I, Tiergartenstr. 17,
D-69121 Heidelberg, Germany

ISBN 3-540-60178-3